НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Інститут прикладного системного аналізу
Кафедра математичних методів системного аналізу

«До захисту допущено»
В.О. завідувача кафедри
_____ О.Л. Тимощук

Дипломна робота

на здобуття ступеня бакалавра

з напряму підготовки 120 Комп'ютерні науки

на тему: «Ансамбль моделей для прогнозування повернення кредитів на основі інформації про поведінку користувача на сайті»

Виконав:
студент IV курсу, групи КА-55
Міщук Андрій Романович _____

Керівник:
професор кафедри ММСА
Мілявський Юрій Леонідович _____

Консультант з економічного розділу:
доцент, к.е.н. Шевчук О. А. _____

Консультант з нормоконтролю:
доцент, к.т.н. Коваленко А.Є. _____

Рецензент:

_____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2019 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут прикладного системного аналізу

Кафедра математичних методів системного аналізу

Рівень вищої освіти – перший (бакалаврський)

Напрям підготовки (програма професійного спрямування) – 120

Комп’ютерні науки (Системи штучного інтелекту)

ЗАТВЕРДЖУЮ

В.О.Завідувача кафедри

_____ О.Л. Тимощук

«___»_____20__ р.

ЗАВДАННЯ

на дипломну роботу студенту

Міщуку Андрію Романовичу

1. Тема роботи «Моделі, методи та короткострокове прогнозування демографічних процесів в Україні», керівник роботи Мілявський Юрій Леонідович, доцент, затверджені наказом по університету від «__»_березня р. №__

2. Термін подання студентом роботи _____

3. Вихідні дані до роботи

4. Зміст роботи

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо)

_____

_____

6. Консультанти розділів роботи*

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Економічний	Шевчук О. А., доцент		

7. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка

Студент А. Р. Міщук

Керівник роботи Ю. Л. Мілявський

* Якщо визначені консультанти. Консультантом не може бути зазначено керівника дипломної роботи.

РЕФЕРАТ

Дипломна робота: 88 с., 22 рис., 8 табл., 2 додатки, 16 джерел.

АНСАМБЛЬ МОДЕЛЕЙ, БУСТИНГ, РЕНДОМ ФОРЕСТ, ГЛИБИННЕ НАВЧАННЯ

Фінансова сфера завжди була одним з основних споживачів інформаційних технологій, зокрема, систем побудови прогнозів. Досить поширеними на сьогодні є різноманітні методи передбачення для фільтрації проблемних клієнтів. Зазвичай для такого аналізу використовується приватна інформація про користувача. Перевагою даної роботи є те, що в якості вхідних даних використовується інформація про поведінку користувача на сайті, що не є приватною. Проте, дані такого типу мають недолік - вони є недостатньо репрезинтативними для застосування в класичних моделях, тому в цій роботі використано методи ансамблювання слабких моделей з метою одержання сильної на їх основі.

Об’єкт дослідження – платоспроможність позичальника.

Предмет долідження – ансамбль моделей для прогнозування дефолту позичальника.

Методи дослідження – техніки ансамблювання беггінг, бустинг, бутстрапінг, стекінг.

ABSTRACT

Thesis contains: 88 p., 22 Fig., 2 Append., 8 Tabl.,16 sources.

The theme: “Prediction of loans repayment based on information about clients behaviour on site”.

MODEL ENSEMBLE, BOOSTING, RANDOM FOREST, DEEP EDUCATION

The financial sphere has always been one of the main consumers of information technology, in particular, systems for forecasting. Today, there are various methods for predicting the filtering of problem clients. Usually, private information about the user is used for such analysis. The advantage of this work is that the input data uses information about the behavior of the user on a site that is not private. However, the data of this type has a drawback - it is not sufficiently representative for use in classical models, so in this work the methods of the assembly of weak models in order to obtain a strong on their basis were used.

The object – client`s solvency.

The subject – model ensemble for customer default prediction.

Methods – bagging, boosting, bootstrapping, stacking model ensemble technics.

ЗМІСТ

ВСТУП

З розвитком інформаційних технологій поступово розширюється сфера їх застосування. Сучасні системи маніпулювання даними здатні забезпечувати майже онлайнові процеси, що дає змогу інтегрувати та застосовувати алгоритми прогнозування ще ближче до користувача. Особливо чутливою до якості прогнозу є фінансова галузь застосування, що використовує різні методи передбачення для фільтрації проблемних клієнтів. Зазвичай для такого аналізу використовується приватна інформація про користувача. Перевагою даної роботи є те, що в якості вхідних даних використовується інформація про поведінку користувача на сайті, що не є приватною. Також для побудови якісної моделі використовується ансамблювання кількох моделей.

Об’єкт дослідження – платоспроможність позичальника.

Предмет долідження – ансамбль моделей для прогнозування дефолту позичальника.

Методи дослідження – техніки ансамблювання беггінг, бустинг, бутстрапінг.

В результаті дослідження була сформована:

1. Дослідження популярних методів ансамблювання моделей
2. Підготовка даних та генерація вхідних даних
3. Розробка базової моделі
4. Ансамблювання моделей одним з відомих методів

Мета роботи - розробка алгоритму прогнозування повернення позики на основі даних про поведінку користувача на сайті. В якості методу дослідження обрано побудову ансамблю моделей.

Пояснювальна записка складається з чотирьох розділів. В першому розділі здійснюється огляд деяких алгоритмів машинного навчання, методів генерації змінних, методів поділу вибірки на тренувальну та тестову, існуючих

методів побудови ансамблів моделей машинного навчання. Другий розділ описує ключові характеристики якості моделей, вибір метрики, підбір базових(слабких) моделей, підбір параметрів для базових моделей, розділення датасету на тренувальну та тестову вибірки та генерацію вхідних змінних на основі поведінкових даних клієнта на сайті. В третьому розділі описується процес розробки ансамблю моделей для прогнозування повернення кредитів. Четвертий розділ містить аналіз економічної частини роботи.

РОЗДІЛ 1 ОГЛЯД ІСНУЮЧИХ МЕТОДІВ ПОБУДОВИ АНСАМБЛІВ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

1.1 Огляд поняття зсуву та дисперсії. Баланс зсуву та дисперсії.

Концептуальне визначення:

- Помилка зміщення - помилка, що приймається як різниця між очікуваним (або середнім) прогнозом нашої моделі і правильним значенням, яке ми намагаємося передбачити. При повторенні процесу побудови моделі більше одного разу: кожен раз, коли ви збираєте нові дані і виконуєте новий аналіз, створюючи нову модель. Внаслідок випадковості базових наборів даних, отримані моделі матимуть ряд прогнозів. Зсув означатиме наскільки далеко ці прогнози передбачають від правильного значення.
- Помилка, обумовлена дисперсією - помилка, що приймається як варіабельність прогнозування моделі для даної точки даних. Дисперсія показує наскільки прогнози для даної точки варіюються між різними реалізаціями моделі.

Ми можемо створити графічну візуалізацію зміщення і дисперсії за допомогою діаграми. Уявіть, що центром мішені є модель, яка чудово передбачає правильні значення. Коли ми віддаляємося від центру, наші прогнози все гірше і гірше. Уявіть, що ми можемо повторити весь процес побудови моделі, щоб отримати декілька окремих попадань в ціль. Кожна точка являє собою індивідуальну реалізацію нашої моделі, враховуючи випадковість варіабельності даних, які ми збираємо. Іноді ми отримуємо гарний розподіл на навчальних даних, тому ми дуже добре прогнозуємо, і ми близькі до центру, а іноді наші навчальні дані можуть бути повними відхилень або нестандартних значень, що призводить до бідніших прогнозів. Ці різні реалізації призводять до розкиду попадань в ціль[14].

На рисунку 1 розглянуто чотири різних випадки, що представляють собою комбінації високого і низького зміщення і дисперсії.

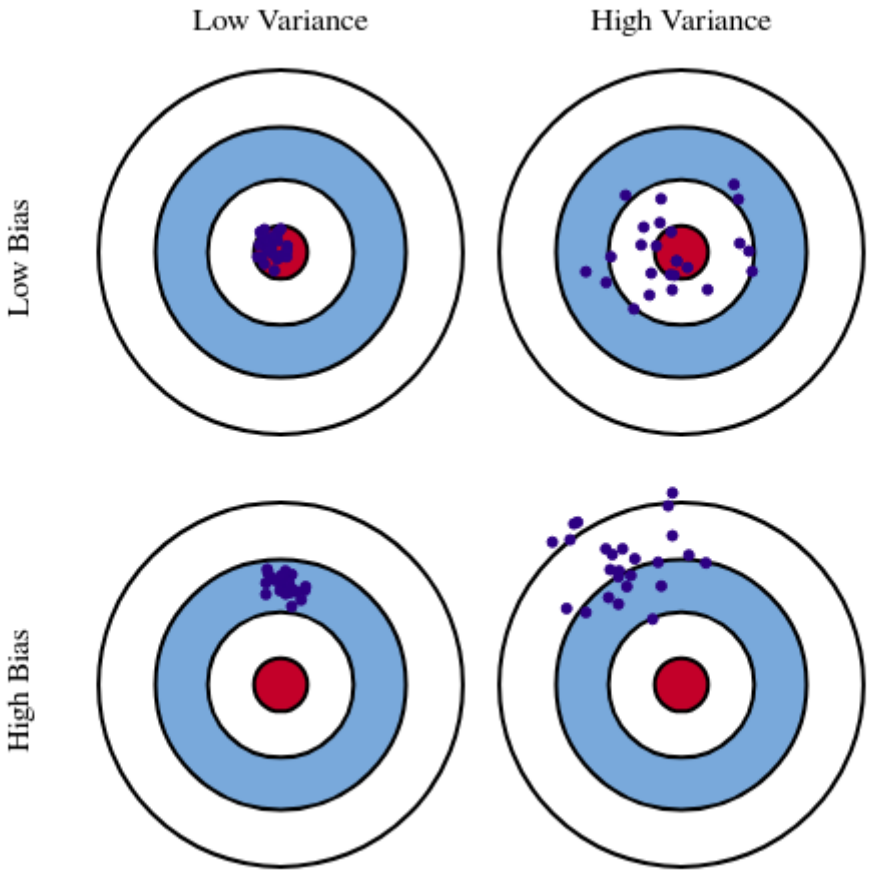


Рисунок 1 - Діаграми комбінації високого і низького зміщення і дисперсії

На вищенаведеній схемі центр мішені є моделлю, яка чудово прогнозує правильні значення. Коли ми відходимо від центру, наші прогнози стають все гірше і гірше. Ми можемо повторити наш процес побудови моделі, щоб отримати окремі попадання в ціль.

У машинному навчанні з учителем, underfitting відбувається коли модель не в змозі охопити основний патерн даних. Ці моделі, як правило, мають високе зміщення і низьку дисперсію. Це відбувається, коли у нас дуже мало даних для побудови точної моделі або коли ми намагаємося побудувати

лінійну модель з нелінійними даними. Крім того, ці моделі занадто прості для захоплення складних моделей даних, таких як лінійна і логістична регресія.

У машинному навчанні з учителем, перенавчання відбувається, коли наша модель фіксує шум разом з базовим шаблоном у даних. Це відбувається, коли ми навчаємо нашу модель забагато над зашумленим набором даних. Ці моделі мають низьке зміщення і високу дисперсію. Ці моделі дуже складні, як дерева рішень, які схильні до перенавчання. На рисунку 2 зображено візуалізацію перенавчання, недостатнього навчання на збалансованого навчання.

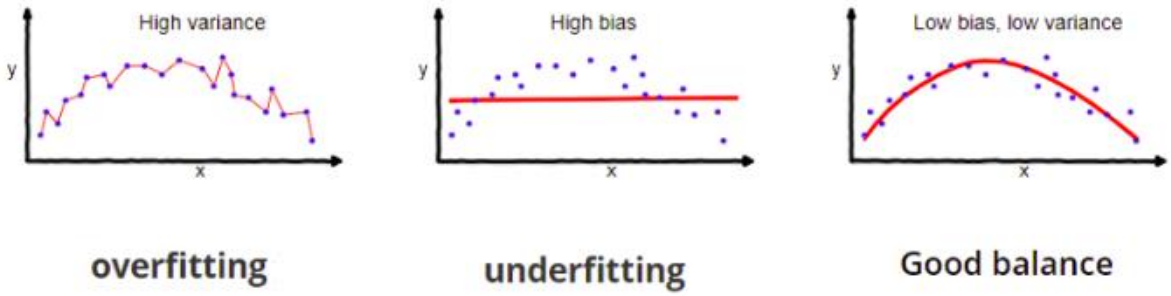


Рисунок 2 - Візуалізація перенавчання та недостатнього навчання

Насправді, боротьба з зсувом та дисперсією - це боротьба з overfit та underfit(надмірним та недостатнім навчанням). Зсув зменшується, і дисперсія збільшується по відношенню до складності моделі. Оскільки все більше і більше параметрів додаються до моделі, то складність моделі зростає, і дисперсія стає нашою першочерговою проблемою, в той час як зсув постійно падає. Наприклад, якщо до лінійної регресії додати більше поліноміальних термінів, то більша складність отриманої моделі буде 3. Іншими словами, зміщення має негативну похідну першого порядку у відповідь на складність моделі 4, тоді як дисперсія має позитивний нахил. На рисунку 3 зображено залежність складності моделі, зсуву та дисперсії.

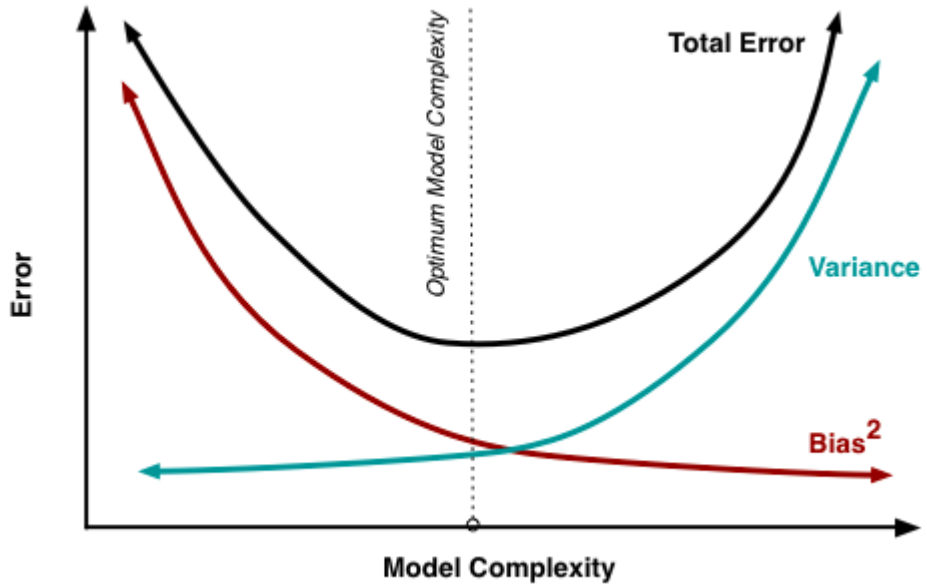


Рисунок 3 - Зсув зменшується, і дисперсія збільшується по відношенню до складності моделі

Розуміння зміщення і дисперсії є критично важливим для розуміння поведінки моделей прогнозування, але в цілому те, що дійсно важливо, є загальною помилкою, а не конкретною декомпозицією.

1.2 Random forest

Random Forest (RF) - це один з багатьох алгоритмів машинного навчання, що використовуються для навчання з учителем, це означає, що можна навчитися з позначених даних і робити прогнози на основі вивчених моделей. RF можна використовувати як для класифікації, так і для регресійних завдань[2].

RF базується на деревах рішень. У машинному навчанні дерева рішень є методом створення моделей прогнозування. Їх називають деревами прийняття рішень, оскільки передбачення слідує за кількома гілками рішення “якщо… тоді…”, поділеним на гілки дерева. Якщо уявити собі, що ми

починаємо з вибірки, для якої ми хочемо передбачити клас, ми почнемо внизу дерева і рухаємося вгору по стовбуру, поки не прийдемо до першого відділення.

Цей поділ можна розглядати як функцію в машинному навчанні, скажімо, нехай це буде «вік»; тепер ми приймемо рішення про те, яка гілка наслідуватиме: «якщо наш зразок має вік більше 30, продовжуйте ліворуч, а далі продовжуйте праворуч». Це ми будемо робити, поки не прийдемо до наступної гілки і не повторимо той самий процес прийняття рішень, поки не буде більше гілок перед нами. Ця кінцева точка називається листом, а в деревах рішень - кінцевий результат: прогнозований клас або значення.

У кожній гілці знайдено порогові значення, які найкраще розділяють дані (що залишилися).

Самі дерева рішень дуже легко уявити і зрозуміти, оскільки вони слідують методу прийняття рішень, який дуже схожий на те, як приймаємо рішення ми: ланцюгом простих правил. Проте, вони не дуже міцні, тобто вони не добре узагальнюють нові зразки даних. Це і є точкою входу для Random Forests.

RF робить прогнози шляхом комбінування результатів з багатьох окремих дерев рішень - тому ми називаємо їх лісом дерев рішень. Оскільки RF поєднує в собі декілька моделей, вона підпадає під категорію ансамблевого навчання. На рисунку 4 зображено структуру алгоритму рандом форест.

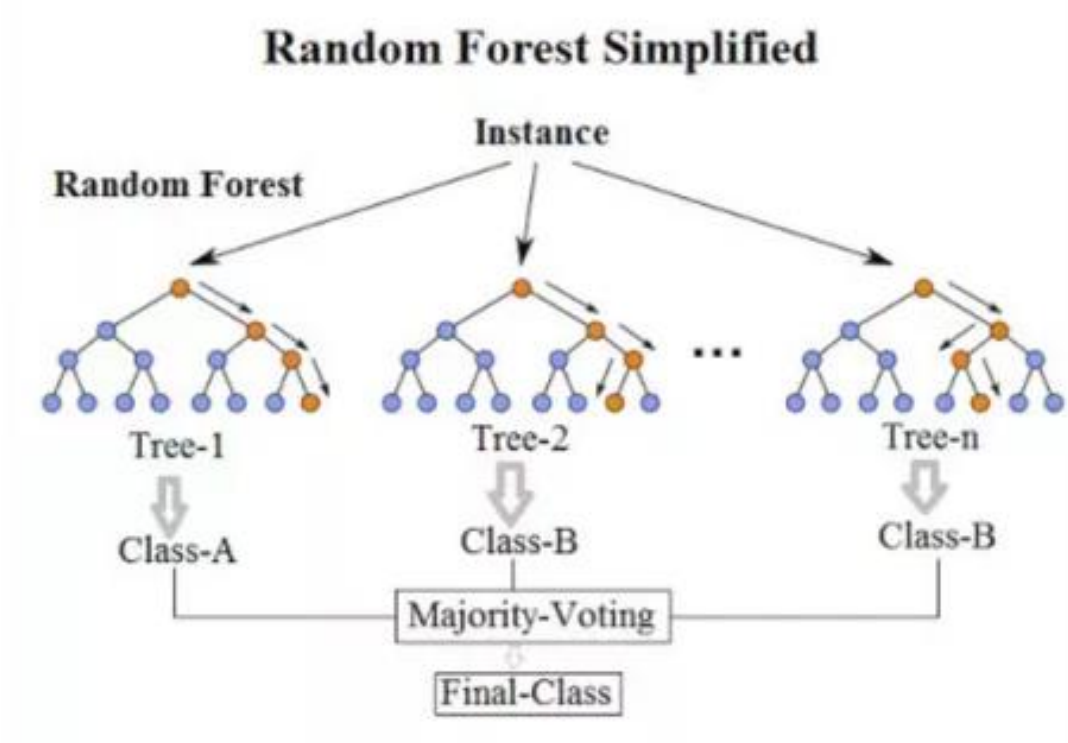


Рисунок 4 - RF робить прогнози шляхом комбінування результатів з багатьох окремих дерев рішень

Гіперпараметри - це аргументи, які можна встановити перед тренуванням і які визначають, як проводиться навчання. Основними гіперпараметрами в Random Forest є:

- Кількість дерев рішень, які необхідно об'єднати
- Максимальна глибина дерев
- Максимальна кількість ознак, що розглядаються при кожному розбитті
- Чи виконується bagging/bootstrapping з або без заміни

Переваги Random Forests полягають у тому, що вони є відносно швидким і потужним алгоритмом навчання класифікації та регресії. Розрахунки можуть бути паралелізовані і добре виконуватися при багатьох

задачах, навіть при малих наборах даних, а вихідні дані повертають ймовірності прогнозування.

Недоліки Random Forests полягають у тому, що вони є чорними ящиками, що означає, що ми не можемо інтерпретувати рішення, прийняті моделлю, тому що вони занадто складні. RF також дещо схильні до перенавчання, і вони, як правило, погано прогнозують недостатньо представлені класи в незбалансованих наборах даних.

1.3 Gradient boosting

Метод ансамблювання Gradient boosting - це метод, який використовується для вирішення проблем класифікації та регресії. Він складається з ряду комбінацій адитивних моделей (слабке навчання), що оцінюються ітеративно, що призводить до більш сильної моделі навчання (більш сильне навчання). Зазвичай, метод Gradient boosting використовується для моделей дерев рішень, однак будь-яка модель може бути використана в цьому процесі, наприклад, логістична регресія.

Розглядаючи використання дерев рішень для підгонки Gradient boosting, метою кожного відповідного дерева рішень є мінімізація функції втрат, тобто мінімізація функції градієнта об'єктів поточної моделі, але для цього ми можемо мати функції втрат з перевагами і недоліками для кожного типу проблеми. Однією з функцій втрат, що використовуються в алгоритмі, є функція втрати Хубера, в якій вона має перевагу, оскільки вона менш чутлива до викидів, ніж більш відомі функції втрат, такі як квадратична помилка і абсолютна похибка.

Крім вибору функції втрат, необхідно визначити на початку алгоритму гіперпараметри, такі як глибина дерева та мінімальна кількість особин у кожному вузлі, оскільки комплексні дерева можуть генерувати перенавчання і прості дерева не можуть моделювати все. (недостатність навчання) Умова

зупинки алгоритму є одним з гіпер-параметрів, які необхідно враховувати, оскільки послідовна мінімізація залишків коригує навіть випадкові помилки.

Завдяки цьому, мінімізація градієнта призводить до перенавчання навчальних зразків. Щоб визначити критерії зупинки, кількість ітерацій на навчальній вибірці коригується шляхом перевірки мінімальних лишків також у зразках перевірки, також відомих як перехресне підтвердження(cross validation).

Щоб підібрати модель за допомогою методу Gradient Boosting, слід дотримуватися деяких кроків і застережень. Як і будь-який інший метод регресії або машинного навчання, необхідно оцінити деякі вимоги до успіху в оцінці.

Вибірка зразків(Sampling) початкової фази є надзвичайно важливою для процесу. Крос-валідація (K-Fold) широко використовується як найкраща практика в машинному навчанні. Процес складається з розбиття набору даних на K рівних частин. Алгоритм буде підготовлений K раз на підмножині даних K-1, де в якості тестового зразка на кожному K-циклі або ітерації використовується одна окрема K-підгрупа.

Інший широко використовуваний процес відбору зразків - це 60-20-20, де 60% використовується як навчальний зразок, а два зразки по 20% кожен використовують для валідації та тестування. Іншим важливим моментом є баланс щодо цільової змінної, виконані тести показують, що результати, отримані в базах даних, де подія була збалансована в 50% (передискретизація або недостатня дискретизація), мають кращі результати, ніж при використанні фактичної пропорції.

Модель Gradient Boosting для цих випадків має високий ризик перенавчання. Через низьку кількість записів модель може стати дуже специфічною (ідентифікація осіб, а не характеристики) і, таким чином, знизити точність на базі тестування.

Процес коригування моделі полягає в оцінці поведінки змінних, які складають модель, шукаючи найкращу параметризацію і стабільність моделей у різних зразках.

1.4 Поділ вибірки на тестову та тренувальну

Тренувальні дані — підмножина даних для навчання моделі. Тестові дані — підмножина даних для тестування навченої моделі.

При розбитті датасету на тестовий та тренувальний важливо витримувати дві умови:

- Тестовий набір досить великий, щоб дати статистично значущі результати
- Тестовий набір є репрезетативним набору даних в цілому. Іншими словами, не вибирайте тестовий набір з іншими характеристиками, ніж навчальний набір.

Об'єднання передбачень з декількох моделей може призвести до більш стабільних прогнозів, а в деяких випадках - до прогнозів, які мають кращу продуктивність, ніж будь-яка з базових моделей.

Ефективні ансамблі потребують учасників, які суперечать. Кожен учасник повинен мати навички (наприклад, прогнозувати краще, ніж випадково), але в ідеалі добре виконувати різними способами. Технічно можна сказати, що ми віддаємо перевагу членам ансамблю, що мають низьку кореляцію в своїх прогнозах або помилки прогнозування.

Один з підходів до заохочення відмінностей між ансамблями полягає в тому, щоб використовувати один і той же алгоритм навчання на різних наборах навчальних даних. Це може бути досягнуто шляхом неодноразового повторного вибору навчального набору даних, який, у свою чергу, використовується для підготовки нової моделі. Кілька моделей навчаються,

використовуючи дещо інші кроки на навчальних дані, і, у свою чергу, роблять різні помилки і часто більш стабільні і кращі прогнози в поєднанні.

Ми можемо називати ці методи ансамблями вибору даних. Перевагою цього підходу є те, що можуть бути використані методи повторної селекції, які не використовують всі приклади в наборі даних для навчання. Будь-які приклади, які не використовуються для відповідності моделі, можуть бути використані як тестовий набір даних для оцінки помилки узагальнення вибраної конфігурації моделі.

Найбільш популярні методи:

- Випадкові розбиття. Набір даних неодноразово підбирався з випадковим поділом даних на навчальний і тестовий набір.
- k-fold Cross-Validation. Набір даних розбивається на k однакових розмірів набори, k моделі навчаються, і кожному набору дається можливість використовуватись в якості набору, де модель може тренуватись на всіх інших наборах, що залишились.
- Bootstrap Aggregation. Вибіркові зразки збирають із заміною і в якості тестового набору використовують приклади, не включені в даний зразок.

Можливо, найбільш широко використовуваним методом ансамблювання sampling є агрегація bootstrap aggregation, яка частіше називається bagging. Повторна вибірка з заміною дозволяє збільшити різницю у навчальному наборі даних, зсуваючи модель і, у свою чергу, призводить до більшої різниці між прогнозами отриманих моделей. Нестабільність моделі та малий набір тестових даних означають, що ми дійсно не знаємо, наскільки добре ця модель буде працювати на нових даних загалом.

Ми можемо спробувати простий метод sampling, неодноразово генеруючи нові випадкові розбиття набору даних у тренувальних і тестових наборах, і застосовувати для нових моделей. Розрахунок середньої

продуктивності моделі в кожному розбитті дасть кращу оцінку загальної помилки моделі.

Потім ми можемо об'єднати декілька моделей, навчених на випадкових розбиттях, з очікуванням, що продуктивність ансамблю, ймовірно, буде більш стабільною і кращою, ніж у середньої одиночної моделі.

Проблема з повторюваними випадковими розбиттями як методом передискретизації для оцінки середньої продуктивності моделі полягає в тому, що вона оптимістична.

Підхід, розроблений для того, щоб бути менш оптимістичним і широко використовуваним в результаті - це метод k-fold cross-validation.

Метод є менш упередженим, оскільки кожен приклад у наборі даних використовується лише один раз у тестовому наборі даних для оцінки продуктивності моделі, на відміну від випадкових розбиттів на тестування, де приклад може використовуватися для оцінки моделі багато разів.

Процедура має єдиний параметр, який називається k, що відноситься до кількості груп, на які слід розбити даний зразок даних. Середній бал кожної моделі забезпечує менш упереджену оцінку продуктивності моделі. Типове значення для k дорівнює 10.

Обмеженням методів випадкових розбиттів і k-fold cross-validation з точки зору ансамблевого навчання полягає в тому, що моделі дуже схожі.

Метод bootstrap є статистичним методом оцінки величин популяцій шляхом усереднення оцінок з декількох невеликих вибірок даних.

Важливо, що зразки будуються шляхом складання спостережень з великої вибірки даних по одному за один раз і поверненням їх до зразка даних після їх вибору. Це дозволяє даному спостереженню бути включеним у дану невелику вибірку більше одного разу. Цей підхід до вибірки називається вибіркою з заміною.

Метод може бути використаний для оцінки продуктивності моделей. Приклади, не вибрані в даному зразку, можуть бути використані як тестовий набір для оцінки продуктивності моделі.

Bootstrap є надійним методом для оцінки продуктивності моделі. Він страждає трохи від оптимістичного упередження, але часто точний так, як і k-fold cross-validation на практиці.

1.5 AUC та ROC

AUC - ROC крива - це метрика продуктивності для проблеми класифікації при різних параметрах порогів.

ROC являє собою криву ймовірності і AUC являє ступінь або міру сепарабельності.

Вона показує, наскільки модель здатна розрізняти класи. Вище AUC, краще модель прогнозує 0 як 0 і 1 як 1. [13]

Sensitivity і specificity є статистичними характеристиками виконання тесту бінарної класифікації, також відомого в статистиці як класифікаційна функція.

- Sensitivity (true positive rate, the recall) вимірює частку 1 які правильно визначені як такі
- Specificity (true negative rate)вимірює частку 0 які правильно визначені як такі

На рисунку 5 зображено залежності між Sensitivity і specificity.

sensitivity, recall, hit rate, or true positive rate (TPR)

$$\mathrm{TPR} = \frac{\mathrm{TP}}{P} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} = 1 - \mathrm{FNR}$$

specificity, selectivity or true negative rate (TNR)

$$\mathrm{TNR} = \frac{\mathrm{TN}}{N} = \frac{\mathrm{TN}}{\mathrm{TN} + \mathrm{FP}} = 1 - \mathrm{FPR}$$

Рисунок 5 - Sensitivity та Specificity

Крива ROC побудована за допомогою TPR проти FPR, де TPR знаходиться на осі y, а FPR - на осі x.

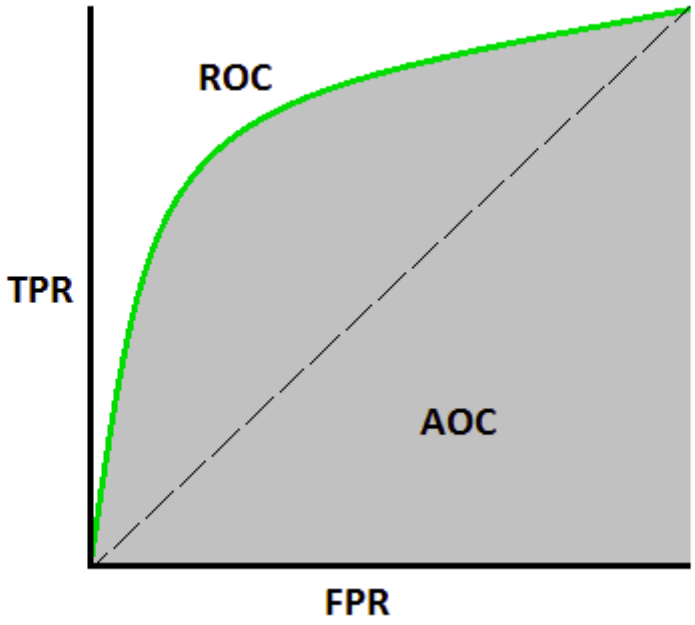


Рисунок 6 - Криві ROC та AUK

Відмінна модель має AUC поблизу 1, що означає, що вона має хорошу здатність розділяти. Слабка модель має AUC поблизу до 0, що означає, що вона має найгірший показник роздільності. Фактично це означає, що він

суперечний. Це передбачає 0 як 1 і 1 як 0. А коли AUC дорівнює 0,5, це означає, що модель не має ніякого класового розподілу.

Як відомо, ROC є кривою ймовірності. Моделюючи розподіли ймовірностей: крива червоного розподілу має позитивний клас , а крива розподілу зеленого - негативний. На рисунку 7 AUC дорівнює 1.

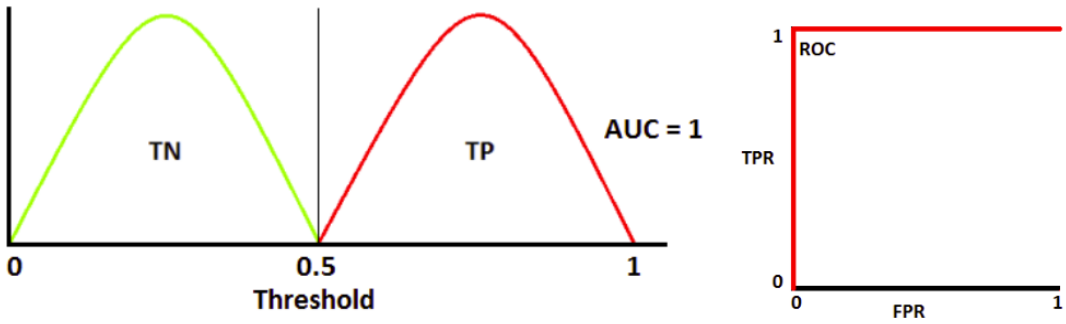


Рисунок 7 - AUC дорівнює 1

Це ідеальна ситуація. Коли дві криві взагалі не перекриваються, модель має ідеальну міру відокремлюваності. Вона прекрасно здатна розрізняти позитивний клас і негативний клас. На рисунку 8 AUC дорівнює 0,7.

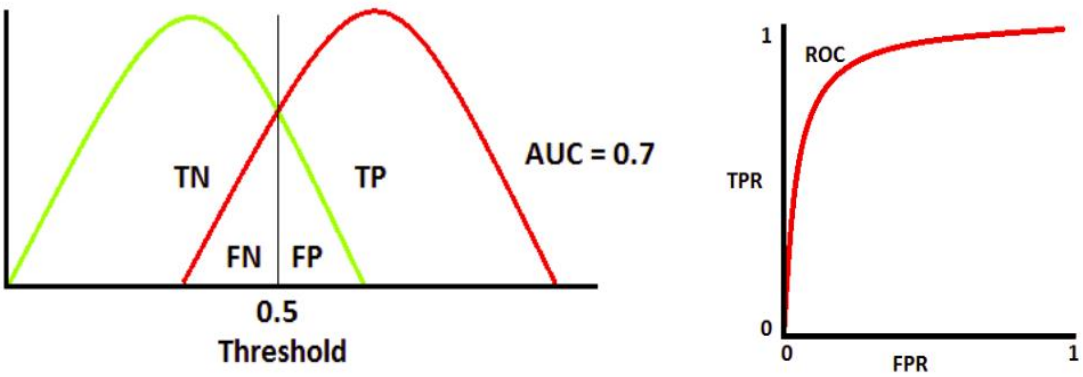


Рисунок - 8 AUC дорівнює 0,7

Коли два розподіли перекриваються, ми вводимо помилки типу 1 і 2 типу . Залежно від порогу ми можемо їх звести до мінімуму або максимізувати. Коли AUC дорівнює 0,7, це означає, що існує 70% шанс, що модель зможе розрізнити позитивний і негативний клас. На рисунку 9 AUC дорівнює 0,5.

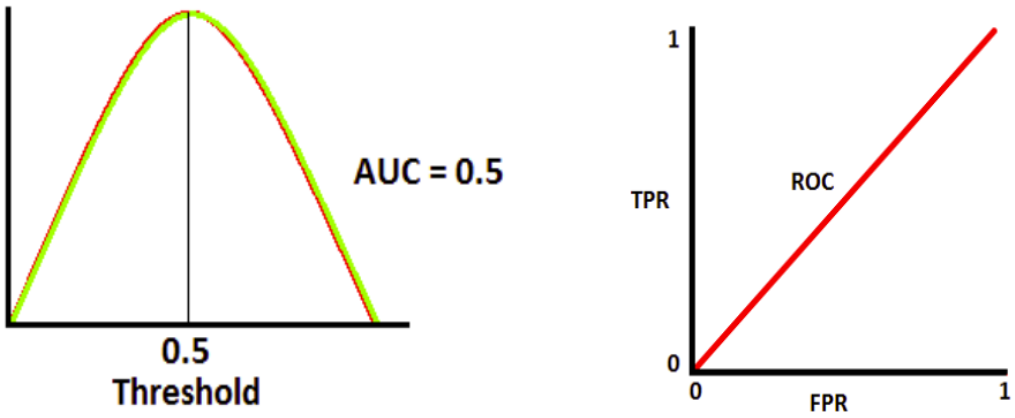


Рисунок 9 - AUC становить приблизно 0,5

Це найгірша ситуація. Коли AUC становить приблизно 0,5, модель не має дискримінаційної здатності розрізняти позитивний клас і негативний клас. На рисунку 10 AUC дорівнює 0.

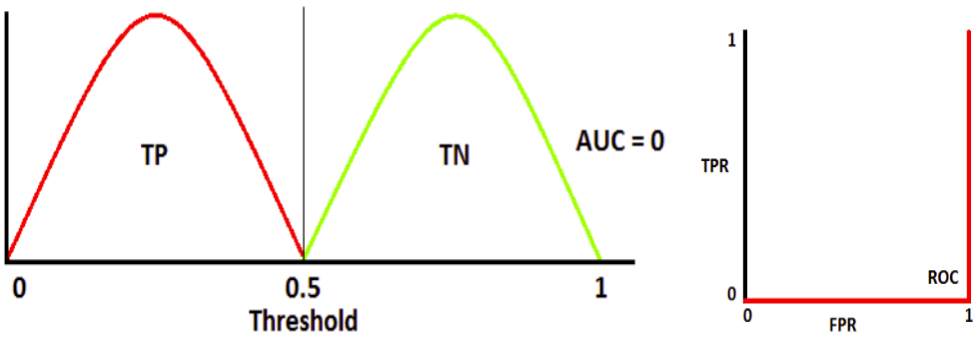


Рисунок 10 - AUC становить приблизно 0

Коли AUC дорівнює приблизно 0, модель насправді є оберненою. Це означає, що модель передбачає негативний клас як позитивний клас і навпаки.

Зв'язок між Sensitivity, Specificity, FPR, Threshold:

- Чутливість і специфічність обернено пропорційні один одному. Тому, коли ми збільшуємо чутливість, специфічність зменшується і навпаки.
- коли ми зменшуємо поріг, ми отримуємо більше позитивних значень, таким чином збільшуючи чутливість і зменшуючи специфіку.
- аналогічно, коли ми збільшуємо поріг, ми отримуємо більше негативних значень, таким чином, отримуємо більш високу специфічність і нижчу чутливість.
- Як відомо, FPR є 1 - специфічністю. Тому, коли ми збільшуємо TPR, FPR також зростає і навпаки.

1.6 Поняття ансамблю моделей

Ансамблювання моделей - це парадигма машинного навчання, основу якої складають методи, в яких кілька моделей (умовно кажучи - “слабких”) тренують задля розв’язання однієї задачі та комбінують, щоб отримати кращі результати. Основною гіпотезою є те, що правильно скомбіновані слабкі моделі дозволяють нам отримати більш точні і/або сильні моделі[1].

В машинному навчанні, не важливо чи ми розв’язуємо задачу класифікації чи регресії, вибір моделі є ключовим складником для одержання гарних результатів. Цей вибір може залежати від багатьох змінних проблеми: кількості даних, розмірності простору, гіпотез розподілу та ін.

Низький зсув(bias) та низька дисперсія(variance), хоч вони зазвичай варіюються в протилежних напрямках, є двома фундаментальними характеристиками, що ми бажаємо отримати від моделі. Дійсно, щоб мати змогу розв’язати задачу, ми хочемо, щоб наша модель мала достатньо ступенів свободи для вирішення основної складності даних, з якими ми працюємо, але ми також хочемо, щоб вона не мала занадто великих ступенів свободи, щоб

уникнути високих відхилень і бути більш надійною. Це і є той самий компроміс зсуву та дисперсії(bias-variance tradeoff).

Ми використовуємо слабкі моделі як базові для ансамблювання більш складних моделей. Переважно, ці базові моделі мають низьку продуктивність поодинці тому, що вони мають високий зсув або зависоку дисперсію.

Тож, ідея ансамблювання полягає в спробах зменшити зсув та/або дисперсію слабких моделей, шляхом комбінування кількох з них з метою створення сильної, що буде достатньо продуктивноюх[7].

Щоб побудувати ансамбль моделей, нам спочатку потрібно обрати базові моделі для агрегації. Якщо ми використовуватимемо в якості бази один алгоритм, натренований різними вибірками, тоді матимемо гомогенний ансамбль моделей. Якщо ж використовуватимемо різні алгоритми навчання - гетерогенний ансамбль.

Важливим нюансом є те, що вибір слабких моделей має бути пов’язаний з способом агрегації. Якщо ми виберемо базові моделі з низьким зсувом, але високою дисперсією, тоді варто використовувати агрегацію, що зменшуватиме дисперсію і навпаки - обираючи базові моделі з низькою дисперсією та високим зсувом, метод агрегації повинен зменшувати зсув.

Це підводить нас до питання як комбінувати ці моделі. Ми можемо зауважити три найбільші типи мета-алгоритмів, що покликані комбінувати слабкі алгоритми:

- bagging, що зазвичай включає гомогенні слабкі моделі, навчає їх паралельно та незалежно одну від одної та комбінує в якесь детерміноване усереднення
- boosting, що зазвичай включає гомогенні слабкі моделі, навчає їх послідовно та адаптивно(підрахунок базової моделі залежить від результату попередніх) та комбінує згідно з детермінованою стратегією

- stacking, що зазвичай включає гетерогенні слабкі моделі, навчає їх паралельно та комбінує їх, навчаючи мета-модель видавати передбачення базоване на передбаченнях слабких моделей.

Грубо кажучи, ми можемо вважати, що bagging переважно фокусується на отриманні ансамблю з нижчою дисперсією, ніж в базових компонентів, boosting та stacking переважно намагаються продукувати сильні моделі з меншим зсувом, ніж в базових компонентів(навіть якщо дисперсія теж зменшиться).

Складно відслідкувати початок історії методів ансамблювання, оскільки базова ідея застосування кількох моделей використовується досить давно. Лише зрозуміло, що хвиля бурхливих досліджень методів ансамблювання 1990х років асоціюється з двома роботами. Перша - це прикладне дослідження[1], яке виявило, що предиктори побудовані з набору класифікаторів часто більш точні за предиктори з одного найкращого класифікатора.

Друга - [2] підтвердила, що слабкі алгоритми можуть бути покращені до сильних, і це доведення переросло в boosting, один з найбільш впливових методів ансамблювання.

1.7 Bootstrapping

Пояснення bootstrapping іноді може бути пропущене багатьма науковцями. Тим не менш, розуміння цього процесу є ключовим, оскільки два з найбільш впливових методів ансамблювання(boosting і bagging) засновані на концепції bootstrapping[6].

У термінах машинного навчання метод bootstrap відноситься до генерації випадкового зразка з заміною. Цей зразок, після заміни, називається повторним. Це дозволяє моделі або алгоритму отримати краще розуміння різних зсувів, відхилень і особливостей, які існують у повторному зразку.

Зразок вибірки даних уможливлює існування різних характеристик в повторному зразку. Це, у свою чергу, вплине на загальне середнє, стандартне відхилення та інші описові показники набору даних. Зрештою, це призводить до розробки більш надійних моделей.

Наведена на рисунку схема зображує кожну популяцію вибірки, що має різні частини.

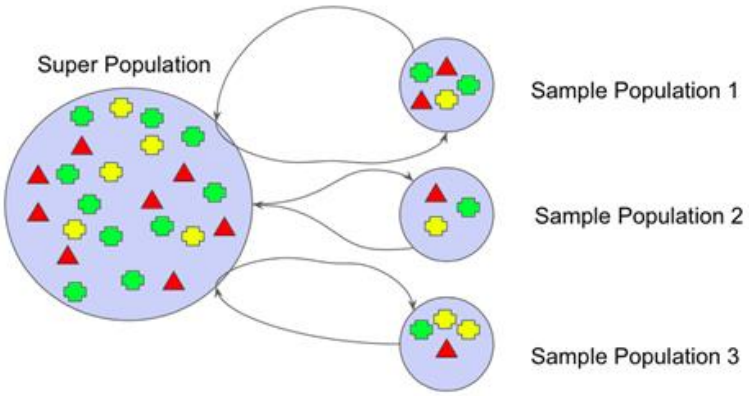


Рисунок 11

Bootstrapping також чудово підходить для наборів даних невеликого розміру, які можуть мати тенденцію до перенавчання. Bootstrapping може бути рішенням у цьому випадку, тому що алгоритми, які використовують bootstrapping, є більш надійними і можуть обробляти нові набори даних залежно від обраної методології (boosting чи bagging).

Метод bootstrap також може перевірити стабільність рішення. Використовуючи кілька наборів зразків даних для тестування моделей можна значно підвищити надійність. У певних випадках, один набір даних вибірки може мати більш середнє, ніж інший або інше стандартне відхилення. Це може

порушити модель, яка була перенавченою і не тестувалася за допомогою наборів даних з різними варіаціями.

Однією з численних причин того, що bootstrapping стає настільки поширеним, є збільшення обчислювальної потужності. Це дозволяє виконувати багато перестановок з різними повторними зразками.

1.8 Bagging

Bagging фактично відноситься до Bootstrap агрегатів[9].

Bagging предиктори є методом генерування декількох версій предиктора і їх використання для отримання агрегованого предиктора.

Bagging допомагає зменшити дисперсію в моделях, які є точними лише за даними, на яких вони навчалися. Ця проблема також відома як перенавчання.

Перенавчання відбувається, коли функція занадто добре підходить до даних. Як правило, це пояснюється тим, що фактичне рівняння занадто складне, щоб врахувати всі точки даних та викиди. На рисунку 12 зображено перенавчання.

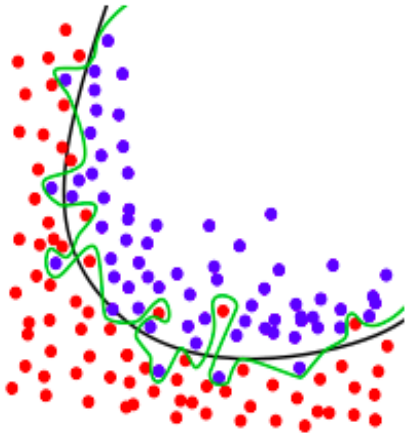


Рисунок 12 - Перенавчання

Іншим прикладом алгоритму, який легко перенавчається, є дерево рішень. Моделі, які розробляються з використанням дерев рішень, вимагають дуже простої евристики. Дерева рішень складаються з набору тверджень if-else, виконаних у певному порядку. Таким чином, якщо набір даних змінено на новий набір даних, який міг би мати деякий зсув або різницю в розповсюдженні базових характеристик у порівнянні з попереднім набором, модель не зможе бути настільки точною, як раніше. Це пояснюється тим, що дані не будуть відповідати моделі.

Bagging обходить проблему перенавчання, створюючи власну дисперсію між даними. Це робиться шляхом вибірки та заміни даних, коли він перевіряє кілька гіпотез (моделей). У свою чергу, це зменшує шум, використовуючи безліч вибірок, які, швидше за все, будуть складатися з даних з різними атрибутами (медіана, середнє тощо).

Як тільки кожна модель розробила гіпотезу, моделі голосують для класифікації або усереднення для регресії. Як показано на рисунку нижче, кожна гіпотеза має таку ж вагу, як і всі інші. На рисунку 13 зображено схему bagging алгоритму.

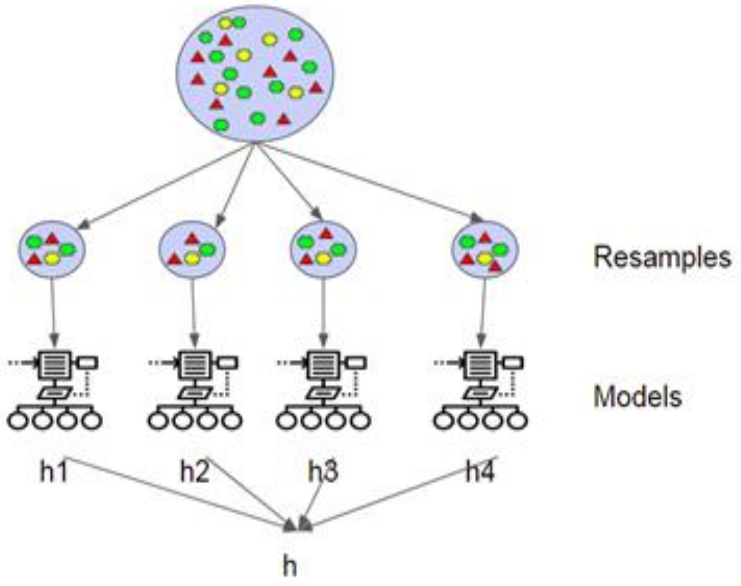


Рисунок 13 - Кожна гіпотеза має таку ж вагу, як і всі інші

По суті, всі ці моделі працюють одночасно і голосують за найбільш точною гіпотезою. Це допомагає зменшити дисперсію, тобто зменшити перенавчання.

1.9 Boosting

Boosting відноситься до групи алгоритмів, які використовують зважені середні значення, щоб слабкі моделі в сильні. На відміну від bagging (де кожна модель працює самостійно, а потім агрегує виходи в кінці без переваги будь-якої моделі), Boosting пов’язує моделі. Кожна модель, що працює, передає наступній, на яких змінних буде зосереджена наступна модель[8].

Boosting також потребує bootstrapping-у. Також, на відміну від bagging-у, boosting зважує зразки даних. Це означає, що деякі зразки будуть працювати частіше, ніж інші. На рисунку 14 зображено схему boosting алгоритму.

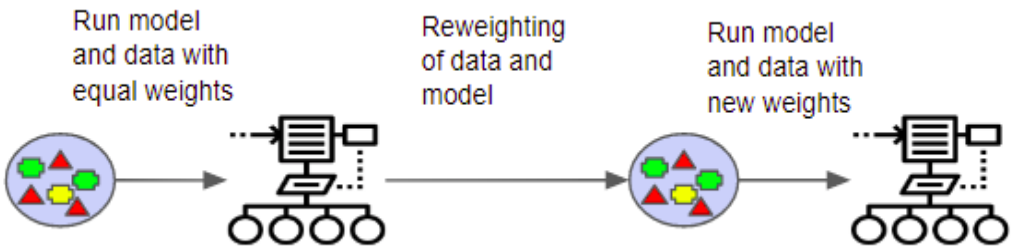


Рисунок 14 - Bootstrapping

При boosting-у виконується кожна модель, вона відстежує, які зразки даних є найбільш успішними і які не є. Наборам даних з найбільш неправильно класифікованими виходами присвоюються більші ваги. Це пояснюється тим, що такі набори даних вважаються більш складними. Таким чином, потрібно більше ітерацій для належного навчання моделі.

Під час фактичної стадії класифікації, boosting відстежує рівень відхилень моделей та присвоює кращі ваги кращим моделям. Таким чином, коли відбувається «голосування», як і в bagging, моделі з кращими результатами мають більший вплив на кінцевий результат.

1.10 Stacking

Stacking в основному відрізняється від bagging і boosting двома нюансами. По-перше, stacking часто розглядає гетерогенні слабкі моделі (комбінуються різні алгоритми навчання), тоді як bagging і boosting розглядають переважно однорідні слабкі моделі. По-друге, stacking допомагає поєднувати базові моделі з використанням метамоделі, тоді як bagging і boosting об'єднують слабкі моделі за детермінованими алгоритмами[10].

Для stacking, ми повинні визначити дві речі для того, щоб побудувати наш ансамбль: L-моделі, які ми хочемо підігнати, і мета-модель, яка їх об'єднує.

Наприклад, для проблеми класифікації ми можемо вибрати як слабких учнів класифікатор KNN, логістичну регресію та SVM, і вирішити вивчити нейронну мережу як мета-модель. Потім нейронна мережа отримає вхідні дані виходів наших трьох слабких моделей і навчиться повертати кінцеві прогнози на її основі[16].

Отже, припустимо, що ми хочемо підігнати ансамбль, що складається з L слабких моделей. Далі ми повинні виконати наступні кроки:

- розділити дані навчання на два набори
- обрати L слабких моделей і надати їм дані з першого набору
- для кожної з L слабких моделей роблять прогнози для спостережень у другому наборі
- натренувати мета-модель на другому наборі, використовуючи прогнози, зроблені слабкими моделями як вхідні дані

На попередніх етапах ми розділили набір на два набори, оскільки прогнози на даних, які використовувалися для підготовки слабких моделей, не є релевантними для навчання мета-моделі. Таким чином, очевидний недолік цього розбиття нашого набору даних полягає в тому, що у нас є лише половина даних для навчання базових моделей, а половина даних для навчання мета-моделі. На рисунку 15 зображено схему stacking алгоритму.

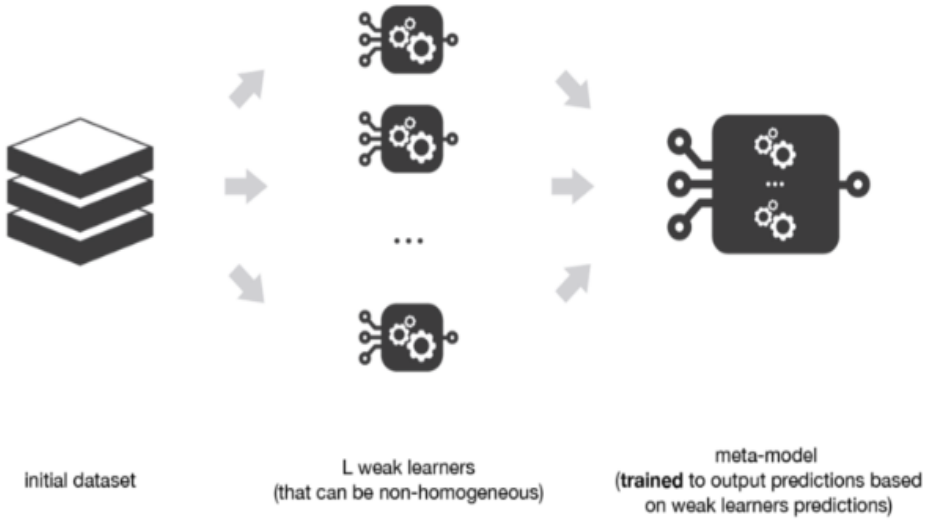


Рисунок 15 - Stacking

Методи ансамблювання зазвичай перевершують одну модель. Boosting та bagging є чудовими методами для зменшення дисперсії. Однак, вони не вирішать всі проблеми, та мають свої власні недоліки. Існують різні причини, чому слід використовувати той чи інший.

Bagging підходить для зменшення дисперсії, коли модель перенавчена. Тим не менш, boosting, ймовірно, буде кращим вибором з двох методів. Це пояснюється тим, що він також чудово підходить для зменшення дисперсії в моделі. З іншого боку, boosting, швидше за все, матиме проблеми з продуктивністю.

Основні моменти:

- ансамблювання моделей - це парадигма машинного навчання, декілька моделей (часто званих слабкими моделями або базовими моделями) навчаються вирішувати одну і ту ж проблему і об'єднуватися, щоб отримати кращі показники
- головна гіпотеза полягає в тому, що якщо ми поєднуємо слабкі моделі, то ми можемо отримати більш точні та / або надійні моделі

- в методі bagging, кілька екземплярів однієї базової моделі навчаються паралельно (незалежно один від одного) на різних зразках bootstrap зразків, а потім агрегуються в якийсь процес "усереднення"
- операція усереднення, що виконується за допомогою базових моделей, дозволяє отримати модель ансамблю з меншою дисперсією, ніж її компоненти: тому базові моделі з низьким зсувом, але з високою дисперсією, добре пристосовані для bagging
- у методах boosting кілька екземплярів однієї базової моделі навчаються послідовно таким чином, що на кожній ітерації спосіб підготовки нинішнього слабкого учня залежить від попередніх слабких учнів і особливо від того, як вони опрацьовують дані
- ця ітераційна стратегія навчання, що використовується в boosting методах, що пристосовується до слабкостей попередніх моделей для підготовки поточної, в основному дозволяє отримати модель ансамблю з меншим зсувом, ніж його компоненти: тому слабкі учні з низькою дисперсією, але високими зсувами добре пристосовані для boosting
- в методах stacking, різні слабкі учні встановлюються незалежно один від одного і мета-модель навчається поверх того, щоб передбачити виходи на основі результатів, що повертаються базовими моделями.

Висновки до розділу 1

В розділі наведено визначення основних об’єктів машинного навчання. Також в розділі розглянуто найважливіші алгоритми та методи ансамблювання моделей.

РОЗДІЛ 2 ОБРОБКА ПОВЕДІНКОВИХ ДАНИХ КЛІЄНТІВ ТА ГЕНЕРАЦІЯ ВХІДНИХ ЗМІННИХ ДЛЯ МОДЕЛЕЙ МАШИННОГО НАВЧАННЯ

2.1 Опис поведінкових даних клієнтів

В якості вхідних даних для розробки ансамблю було використано такі структури:

- список подій(логів), що здійснив користувач на сайті
- дані про стан кредиту з прив’язкою до id кредиту

Опис полів логу з сайту:

- advanceID - ідентифікатор кредиту
- Id - ідентифікатор події
- SessionId - ідентифікатор сесії
- ClientId - ідентифікатор клієнта
- FieldId - ідентифікатор поля на сайті
- Value - значення поля на сайті
- ChangeType - тип події
- Timestamp - час події
- FocusTimestamp - час фокусу на полі
- AvgSpeed - середня швидкість набору(для текстових полів)
- SdSpeed - стандартне відхилення швидкості набору
- AuxText
- IsMobileDevice - чи з мобільного девайсу подія
- Browser – браузер
- SourceType - тип девайсу
- CreateDate
- PageId - версія сайту
- DeviceId - ідентифікатор девайсу

Опис даних про стан кредиту:

- advanceID - ідентифікатор кредиту
- LoanNum
- ProductName
- AGDateCreated
- Fails35NoPayIn90 - чи є 35 днів прострочених виплат впродовж перших 90 днів

- LoanWriteOff - борг вважається списаним через малоймовірність повернення

2.2 Підготовка даних. Генерація вхідних змінних(features). Вибір базової моделі та поділ датасету.

Найважливішим складником будь-якої моделі є дані. Оскільки поведінкові дані з сайту зняті в близьких до реальності умовах(досить складні та “брудні”) - вони потребують обробки. В якості інструментів для обробки датасету було використано:

- Python 3.6 Interpreter
- Python IDE JetBrains PyCharm Community Edition
- Pandas
- Numpy
- Sklearn
- Skipy

Перш за все дані було зведено в єдину структуру, здійснивши OUTER JOIN по advanceID. Далі з отриманої таблички зробили вужчу, відсіявши деякі колонки, що вважались нецікавими.

Отримано структуру:

- advanceID
- FieldId
- Value
- ChangeType
- Timestamp
- FocusTimestamp
- AvgSpeed
- SdSpeed
- AuxText
- IsMobileDevice
- Browser
- SourceType

- CreateDate
- PageId
- DeviceId
- Fails35NoPayIn90

Змінні видобувались шляхом групування по advanceID та агрегації інформаційних полів. Оскільки дані були досить “брудними”(багато NaN/INFINITE значень, порушена послідовність колонок) - була проведена додаткова обробка цих кейсів. В якості вхідних змінних було відібрано такі агрегати:

- preferredFieldId
- preferredChangeType
- preferredWeekDay
- countMonTimestamp
- countTueTimestamp
- countWedTimestamp
- countThuTimestamp
- countFriTimestamp
- countSatTimestamp
- countSunTimestamp
- maxAvgSpeed
- minAvgSpeed
- avgAvgSpeed
- maxSdSpeed
- minSdSpeed
- avgSdSpeed
- preferredIsMobile
- preferredDevice
- preferredBrowser
- preferredSourceType

- preferredPageId
- count121PageId
- count9PageId
- countUniqueDeviceId
- eventCount

На рисунку зображено матрицю кореляційних коефіцієнтів Пірсона для вхідних змінних.

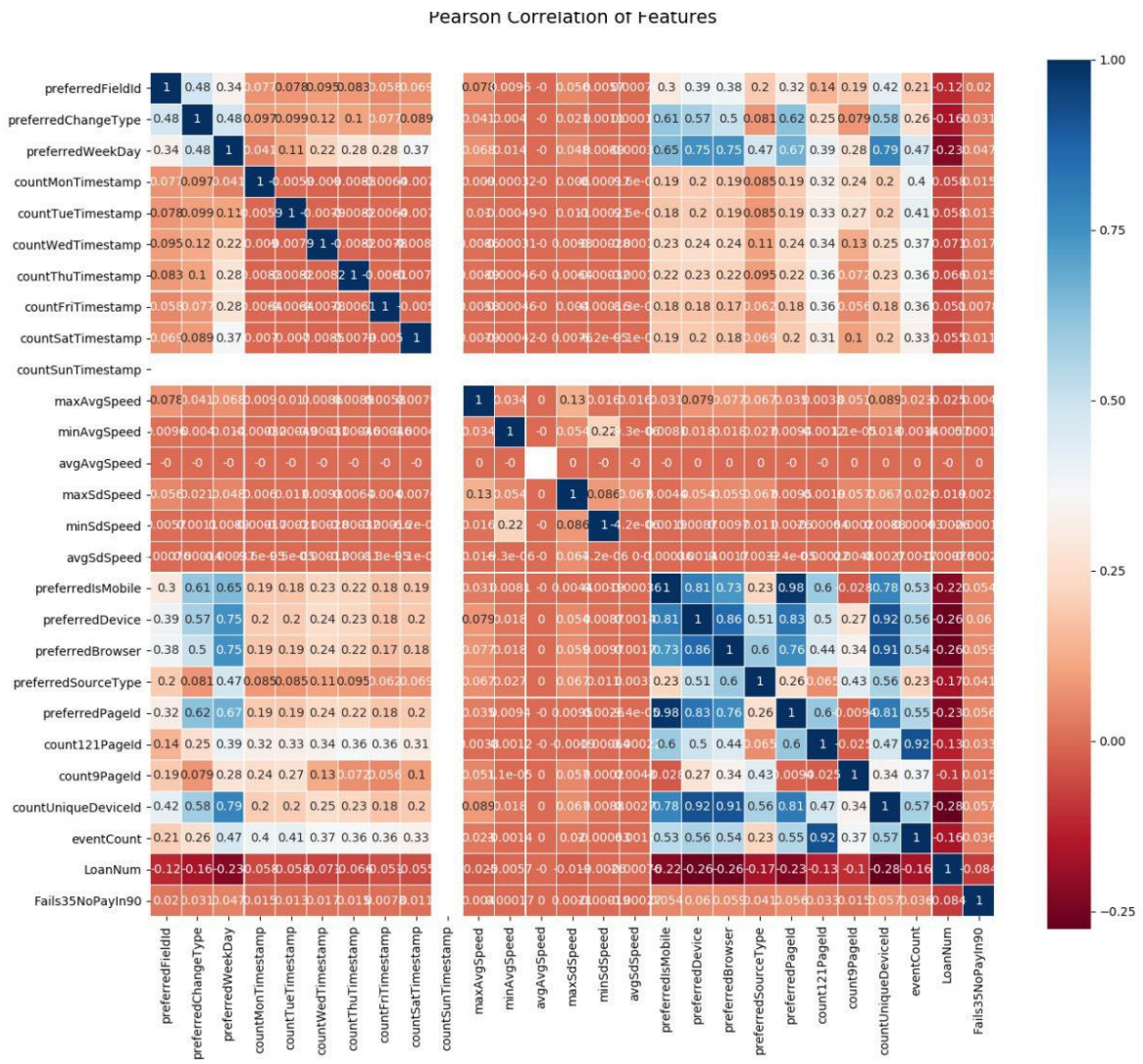


Рисунок 16 – Матриця кореляційних коефіцієнтів Пірсона для вхідних змінних

Одна річ, яку може сказати нам матриця кореляції Пірсона, полягає в тому, що існує не так багато змінних, які сильно корелюють між собою. Це добре з точки зору подачі цих змінних у нашу навчальну модель, оскільки це означає, що в нашому навчальному наборі немає зайвих або надлишкових даних, і ми раді, що кожна змінна містить певну унікальну інформацію.

В якості цільової функції можна було обрати два поля: Fails35NoPayIn90 або LoanWriteOff. Оскільки в LoanWriteOff досить рідко траплялись позитивні
значення та зміст Fails35NoPayIn90 краще підходить під нашу задачу - було обрано Fails35NoPayIn90.

2.2 Поділ датасету

Для поділу датасету використано функції для роботи з алгоритмами та датасетом з бібліотеки sklearn. Як згадувалося в попередньому розділі, stacking використовує прогнози базових класифікаторів як вхідні дані для навчання моделі другого рівня. Проте неможливо просто навчити базові моделі на повних навчальних даних, генерувати прогнози на повному тестовому наборі, а потім виводити їх для другого рівня підготовки. Це спричиняє ризик того, що прогноз вашої базової моделі вже "бачив" тестовий набір і, отже, перенасичується під час подачі цих прогнозів. Тому датасет було розділено на тестовий та тренувальний з допомогою крос-валідатора KFold.

Мета крос-валідації полягає у визначенні набору даних для тестування моделі на етапі навчання (тобто набору даних перевірки), щоб обмежити проблеми, такі як перенавчання, недостатнє навчання та отримати уявлення про те, як модель буде узагальнюватися до незалежного набору даних . Важливо, щоб валідація і тренувальний набір витягувалися з одного розподілу, інакше це погіршить ситуацію.

Висновки до розділу 2

Підготовка вхідних змінних безумовно є найскладнішим етапом розробки моделі. В розділі детально розглянуто вхідні дані та природу їхнього походження, процес вибору змінних, залежності між змінними, спосіб поділу датасету на тренувальний та тестовий.

РОЗДІЛ 3 РОЗРОБКА АНСАМБЛЮ МОДЕЛЕЙ ДЛЯ ПРОГНОЗУВАННЯ ПОВЕРНЕННЯ КРЕДИТІВ

3.1 Вибір методу ансамблювання

Загалом, можна подумати про ансамблювання як про використання ваг. Наприклад, ви можете вважати, що деякі моделі є кращими або більш точними, і ви хочете вручну призначити їм більшу вагу. Але ще кращим підходом може бути оцінка цих ваг більш розумно за допомогою іншого рівня алгоритму навчання. Такий підхід називається моделюванням моделей.

Модельне стекування є ефективним методом ансамблювання, в якому передбачення, сформовані за допомогою різних алгоритмів машинного навчання, використовуються як вхідні дані в алгоритмі навчання другого рівня. Цей алгоритм другого рівня підготований для оптимального поєднання модельних прогнозів для формування нового набору прогнозів. Наприклад, коли лінійна регресія використовується як моделювання другого шару, вона оцінює ці ваги шляхом мінімізації помилок з найменшими квадратами. Однак моделювання другого шару не обмежується лише лінійними моделями; зв'язок між предикторами може бути більш складним, відкриваючи двері для використання інших алгоритмів машинного навчання.

В якості базових моделей першого шару обрано:

- Random Forest класифікатор
- Extra Trees класифікатор
- AdaBoost класифікатор
- Gradient Boosting класифікатор
- Support Vector Machine

3.2 Параметри моделей базового шару

Для завершеності формування шару базових моделей розглянемо параметри.

n_jobs - Кількість ядер, що використовуються для тренувального процесу. Якщо встановлено значення -1, використовуються всі ядра

n_estimators - Кількість дерев класифікації в моделі навчання (встановлено на 10 за умовчанням)

max_depth - Максимальна глибина дерева, або наскільки повинен бути широкий вузол. Якщо встановлено занадто велике число, це спричинить ризик перенавчання, оскільки дерево буде рости занадто глибоко

verbose - Контролює, чи потрібно виводити будь-який текст під час процесу навчання. Значення 0 пригнічує весь текст, а значення 3 виводить процес навчання дерева при кожній ітерації.

3.3 Зв’язок між першим та другим шаром

Підготувавши наші базові моделі першого шару, ми тепер можемо готувати тренувальні та тестові дані для введення в наші класифікатори, генеруючи масиви з допомогою NumPy з їх оригінальних датафреймів.

Далі ми подаємо тренувальні та тестові дані до наших 5 базових класифікаторів і використовуємо функцію прогнозування Out-of-Fold(KFold), яку ми визначили раніше, щоб сформувати наші передбачення першого рівня.

Наступним кроком ми дістаємо з базових моделей вектори важливості змінних та будуємо з них новий датафрейм.

Отримавши наші прогнози першого рівня, можна вважати, що це, по суті, будує новий набір функцій, які будуть використовуватися як навчальні дані для наступного класифікатора. Таким чином, маємо наші нові стовпці

прогнозів першого рівня з наших попередніх класифікаторів, і тренуємо наступний класифікатор на цьому.

3.4 Побудова моделі другого рівня

В якості бібліотеки другого шару ми використовуватимемо XGBoost. Як правило, XGBoost є швидким. Дійсно швидким, у порівнянні з іншими реалізаціями GradientBoosting. Ми використовуємо XGBClassifier і тренуємо його на тренувальних прогнозах моделей першого рівня.

Висновки до розділу 3

З допомогою сучасних засобів сьогодні досить легко використовувати різні алгоритми машинного навчання. Проте, ансамблювання вимагає більшого стеку навичок та технік. В даному розділі здійснено опис техніки побудови стекінг ансамблю моделей.

Застосування стекових моделей до реальних проблем великих даних може забезпечити більшу точність прогнозування та надійність, ніж окремі моделі.

РОЗДІЛ 4 ФУНКЦІОНАЛЬНО-ВАРТІСНИЙ АНАЛІЗ ПРОГРАМНОГО ПРОДУКТУ

4.1 Вступна частина

Даний розділ присвячений функціонально-вартісному аналізу програмного продукту — ансамбль моделей для передбачення повернення кредитів користувача на основі поведінкових даних з сайту.

Функціонально-вартісний аналіз — це метод комплексного техніко-економічного дослідження об’єкта з метою розвитку його корисних функцій при оптимальному співвідношенні між їхньою значимістю для споживача і витратами на їхнє здійснення. Є одним з основних методів оцінки вартості науково-дослідної роботи, оскільки ФВА враховує як технічну оцінку продукту, що розробляється, так і економічну частину розробки. Крім того, даний метод дозволяє вибрати оптимальний, як з погляду розробника, так і з точки зору покупця варіант розв’язання будь-якої задачі, а також дозволяє оптимізувати витрати й час виконання робіт. Зниження витрат виробництва треба починати з аналізу властивостей виробу, що використовуються, а також технічних функцій його складових частин.

У даній роботі проводиться економічний, техніко-економічний аналіз програмного продукту, основним завданням якого є аналіз даних, а також виключення зайвих функцій, що дозволяє пришвидшити процес прийняття рішень, забезпечивши при цьому якісні та достовірні результати.

4.2 Постановка завдання техніко-економічного дослідження

Розроблювальний програмний продукт (ПП) є ансамблем моделей для передбачення повернення кредитів користувача на основі поведінкових даних з сайту.

Задачею проекту було створити ефективну модельзгідно з обраною метрикою.

Технічні вимоги до прподукту:

- простота конфігурування;
- швидкодія при обробці інформації;
- мінімальні затрати на впровадження ПП

Для розробки ПП використовувалось середовище JetBrainsPyCharm 2019.

4.3 Виділення основних функцій

Головна функція F0 – розробка програмного продукту, який дає найменшу похибку при порівняння реальних значень цільової змінної та результуючого значення побудованого прогнозу, у відповідній точці. Виходячи з конкретних цілей, реалізованих програмним засобом, виділимо її основні функції:

F1 – вибір мови програмування;

F2 – вибір методу прогнозування;

F3 – вибір показника гірничопрохідницьких робіт для прогнозування;

F4 – представлення вихідних даних;

Кожна з основних функцій може мати декілька варіантів реалізації.

Функція F1:

а) мова програмування C++;

б) мова програмування Python;

в) мова програмування MatLab.

Функція F2:

а)stacking;

б)boosting;

в) bagging.

Функція F3:

а) швидкість прохідки;

б) швидкість будівництва;

Функція F4:

а) виведення в Excel;

б) виведення в текстовий файл;

в) зображення вихідних даних на графіку.

4.4 Варіанти реалізації основних функцій

Варіанти реалізації основних функцій наведено у морфологічній карті системи на рисунку 17. На основі цієї карти побудовано позитивно-негатвину матрицю варіантів основних функцій, що зображена на таблиці 1.

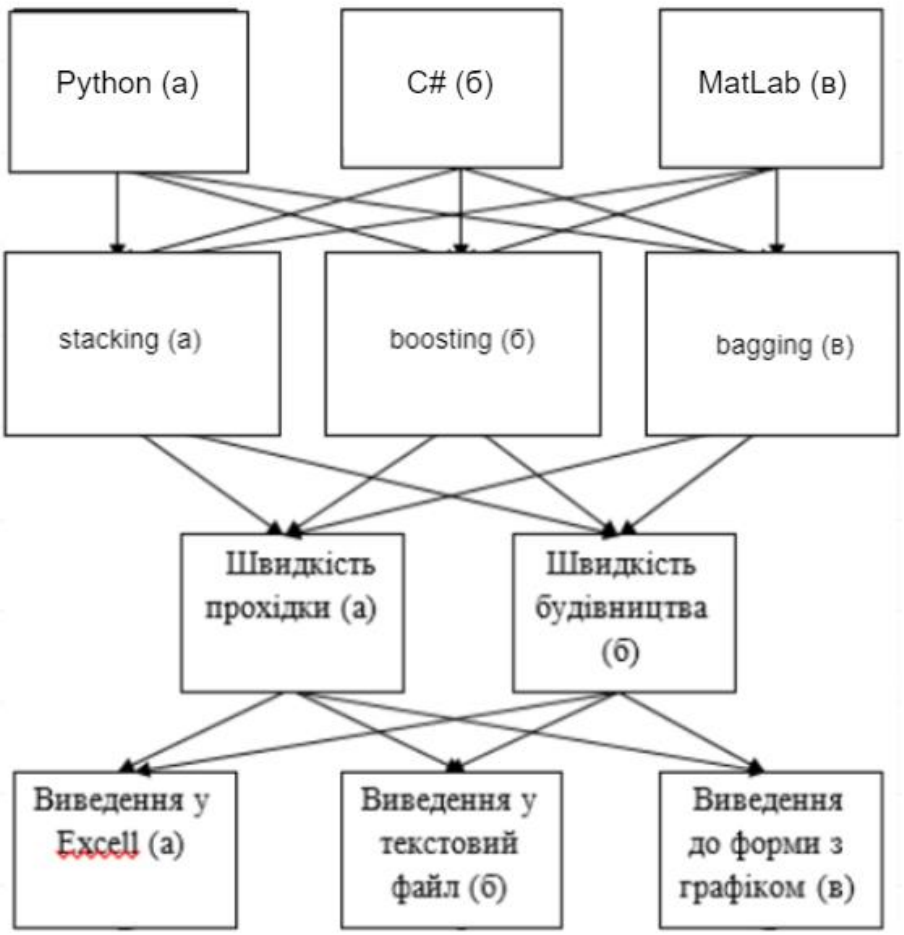


Рисунок 17 – Морфологічна карта

Морфологічна карта відображає всі можливі комбінації варіантів реалізації функцій, які складають повну множину варіантів програмного продукту.

Таблиця 1 – Позитивно-негативна матриця

Основні функції	Варіанти реалізації	Переваги	Недоліки
F1	а	Займає менше часу при написанні коду	Не кросплатформений
	б	Код швидко виконується, кросплатформений	Займає більше часу при написанні коду
	в	Має значну кількість вбудованих математичних фунцій, що значно прискорює написання коду	Не кросплатформений
F2	а	Найкращі результати	Не є найоптимальнішим
	б	Легкість у виконанні, кращі результати, чим при поліноміальному прогноз.	Простота реалізації
	в	Легкість реалізації	Складний при реалізації, затрачений час
F3	а	Краща точність прогнозу	Затрачений час
	б	Легша обробка даних	Менша точність підрахунків

Продовження таблиці 1.

F4	а	Можливість аналізу даних, побудови графіків	Складність реалізації
	б	Простота реалізації	Відсутність можливості одразу почати аналіз данних
	в	Найбільша наочність	Складність реалізації
	б	Стабільний у використанні, проста інтеграція з C++	Складний у створенні

На основі аналізу позитивно-негативної матриці робимо висновок, що при розробці програмного продукту деякі варіанти реалізації функцій варто відкинути, тому що вони не відповідають поставленим перед програмним продуктам задачам.

Функція F1: оскільки розрахунки проводяться з великими об’ємами вхідних даних, то час написання та виконання програмного коду є найважливішими факторами, отже варіант а) та в) потрібно відкинути.

Функція F2: оскільки планується побудувати модель, яка гарно прогнозує при не великих затратах часу, то не має сенсу створювати складний за реалізацією частковий опис, тобто варіанти б) і в) можна відкинути.

Функція F3: оскільки планується розглядати широкий спектр вхідних даних, яким властиві викиди, а також головна мета – це досягнення найточнішої апроксимації, то варіант б) варто відкинути з розгляду.

Функція F4: оскільки варіант виведення даних, є суттєвим для користувача ПП, то можна залишити обидва варіанти, а користувач сам обере як йому зручніше виводити.

Таким чином будемо використовувати наступні варіанти реалізації програмного продукту:

а) F1б – F2а – F3а – F4а;

б) F1б – F2а – F3а – F4в;

в) F1б – F2а – F3б – F4а;

г) F1б – F2а – F3б – F4в.

4.5 Опис параметрів

Маючи вимоги щодо основних функцій, які реалізовуються в програмному продукті, визначають основні параметри виробу, які надалі використовуватимуться для розрахунку коефіцієнта технічного рівня.

Введемо наступні параметри:

X1 – швидкодія мови програмування;

X2 – об’єм пам’яті для збереження даних;

X3 – час обробки даних, залежно від методу прогнозування;

X4 – точність розв’язку;

X5 – потенційний об’єм програмного коду.

X1: відображає швидкодію операцій залежно від обраної мови програмування

X2: відображає об’єм пам’яті в оперативній пам’яті персонального комп’ютера, необхідний для збереження та обробки даних підчас виконання програми.

X3: відображає час, який витрачається на дії залежні від вибору апроксимуючого багаточлену.

X4: показує точність розв’язку, а саме за допомогою вектора похибок;

X5: відображає розмір програмного коду для створення.

4.6 Кількісна оцінка параметрів

Будемо розглядати 3 типи варіантів значення параметрів. Результати наведено в таблиці 2.

Таблиця 2 – Основні параметри програмного продукту

Назва параметра	Умовні позначення	Одиниці виміру	Значення параметра		
			гірші	середні	кращі
Швидкодія мови програмування	X1	Оп/мс	19000	11000	2000
Об’єм пам’яті для збереження даних	X2	Мб	32	16	8
Час обробки даних	X3	мс	800	420	60
Точність розв’язку	X4	доля одиниці	10E-4	10E-5	10E-6
Потенційний об’єм програмного коду	X5	кількість рядків	2000	1500	1000

По даним таблиці 2 побудовано графічні характеристики, зображені на рисунку 18-22.

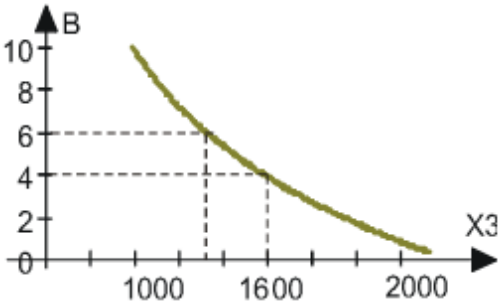


Рисунок 18 – Бальна оцінка швидкодії мови програмування

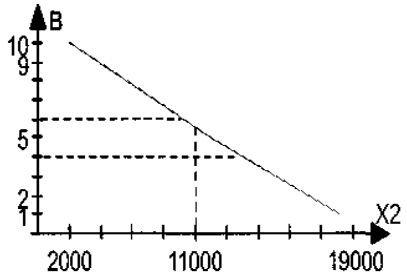


Рисунок 19– Бальна оцінка об’єму пам’яті, що займається даними

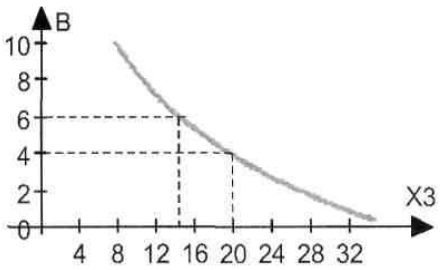


Рисунок 20 – Бальна оцінка часу обробки даних

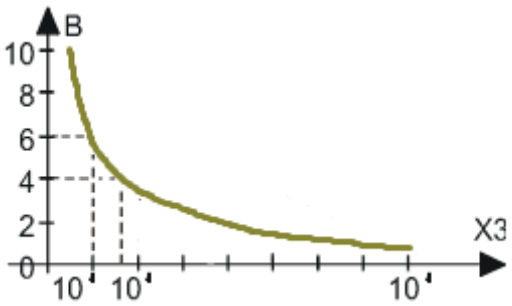


Рисунок 21 – Бальна оцінка точності розв’язку

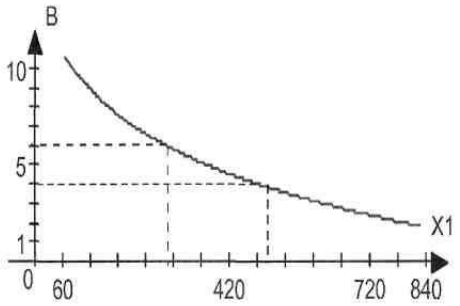


Рисунок 22 – Бальна оцінка об’єму коду програмного продукту

4.7 Аналіз експертного оцінювання параметрів

Кожен експерт оцінює ступінь важливості кожного параметру для конкретно поставленої цілі – розробка програмного продукту, який дає найбільш точні результати при знаходженні апроксимуючого багаточлену. Вага параметрів визначається методом їхнього попарного порівняння, використовуючи результати ранжування експертів. Результати ранжування параметрів зображені на таблиці 3.

Таблиця 3 – Результати ранжування показників

Познач. параметра	Ранг параметра за оцінкою експерта							Сума рангів R_i	Відхилення Δ_i	$\Delta i2$
	1	2	3	4	5	6	7			
X1	4	3	4	4	3	4	3	25	4	16
X2	3	4	3	3	4	3	4	24	3	9
X3	2	2	1	2	1	2	2	12	-9	81
X4	5	5	5	4	5	5	5	34	13	169
X5	1	2	2	2	1	2	1	10	-11	121
	15	16	15	15	14	16	14	105	0	396

Таблиця 4 – Результати ранжування параметрів

Параметри	Експерти							Підсумкова оцінка	Числове значення
	1	2	3	4	5	6	7		
X1,X2	>	<	>	>	<	>	<	>	1,5
X1,X3	>	>	>	>	>	>	>	>	1,5
X1,X4	<	<	<	=	<	<	<	<	0,5
X1,X5	>	>	>	>	>	>	>	>	1,5
X2,X3	>	>	>	>	>	>	>	>	1,5
X2,X4	<	<	<	<	<	<	<	<	0,5
X2,X5	>	>	>	>	>	>	>	>	1,5
X3,X4	<	<	<	<	<	<	<	<	0,5
X3,X5	>	=	<	=	=	=	>	>	1,5
>	>	=	>	>	=	>	>	>	1,5

Розрахуємо коефіцієнт конкордації (узгодженості) експертних оцінок.

Для перевірки степені достовірності експертних оцінок, визначимо наступні параметри:

а) сума рангів кожного з параметрів і загальна сума рангів:

$$R_i = \Sigma_j^N r_{ij} R_{ij} = \frac{Nn(n+1)}{2} = 105 \quad (1)$$

де N – кількість експертів, n – кількість параметрів;

б) середня сума рангів:

$$T = \frac{1}{n} R_{ij} = 21 \quad (2)$$

в) відхилення суми рангів кожного параметра від середньої суми рангів:

$$\Delta_i = R_i - T \quad (3)$$

Сума відхилень по всім параметрам повинна дорівнювати 0;

г)загальна сума квадратів відхилення:

$$S = \sum_{i=1}^{n} \Delta_i^2 = 396 \quad (4)$$

Порахуємо коефіцієнт узгодженості:

$$W = \frac{12S}{N^2(n^3 - n)}, W = \frac{12 * 396}{(7^2 * (5^3 - 5))} = 0{,}80 > W_k = 0{,}67 \quad (5)$$

Ранжування можна вважати достовірним, тому що знайдений коефіцієнт узгодженості перевищує нормативний, котрий дорівнює 0,67.

Скориставшись результатами ранжирування, проведемо попарне порівняння всіх параметрів і результати занесемо у таблицю.

Числове значення, що визначає ступінь переваги *i*–го параметра над *j*–тим, a_{ij} визначається по формулі:

$$a_{ij} = \begin{cases} 1.5, & X_i > X_j \\ 1.0, & X_i = X_j \\ 0.5, & X_i < X_j \end{cases} \qquad (6)$$

З отриманих числових оцінок переваги складемо матрицю A=$\| a_{ij} \|$.

Для кожного параметра зробимо розрахунок вагомості $K_{вi}$ за наступною формулою:

$$\hat{E}_{â^3} = \frac{b_i}{\sum_{i=1}^{n} b_i}, äå\ b_i = \sum_{i=1}^{n} a_{ij} \qquad (7)$$

Відносні оцінки розраховуються декілька разів доти, поки наступні значення не будуть незначно відрізнятися від попередніх (менше 2%).На другому і наступних кроках відносні оцінки розраховуються за наступними формулами:

$$\hat{E}_{â^3} = \frac{b_i'}{\sum_{i=1}^{n} b_i'}, äå\ b_i' = \sum_{j=1}^{n} a_{ij} b_j \qquad (8)$$

Розраховані вагомості параметрів внесені в таблицю 5

Таблиця 5 – Розрахунок вагомості параметрів

Параметри	Параметри					Перша ітер.		Друга ітер.	
	X1	X2	X3	X4	X5	b_i	$К_{вi}$	b_i^1	$\hat{E}_{\hat{a}^3}^{1}$
X1	1,0	1,5	1,5	0,5	1,5	6	0,2	28,5	0,2415254237
X2	0,5	1,0	1,5	0,5	1,5	5	0,28	23	0,1949152542
X3	0,5	1,5	1,0	0,5	1,5	5	0,12	23	0,1949152542
X4	1,5	0,5	1,5	1,0	1,5	6	0,16	29,5	0,25
X5	0,5	0,5	0,5	0,5	1,0	3	0,24	14	0,1186440678
Всього:						25	1	118	1

Як видно з таблиці, різниця значень коефіцієнтів вагомості не перевищує 2%, тому більшої кількості ітерацій не потрібно.

4.8 Аналіз рівня якості варіантів реалізації функцій

Визначаємо рівень якості кожного варіанту виконання основних функцій окремо.

Абсолютні значення параметрів X2(об’єм пам’яті для збереження даних) та X1(швидкодія мови програмування) відповідають технічним вимогам умов функціонування даного ПП.

Абсолютне значення параметра X3 (час обробки даних) обрано не найгіршим (не максимальним), тобто це значення відповідає або варіанту а) 800 мс або варіанту б) 60мс.

Абсолютні значення параметра X4 (точність розвязку) обрані з наступних міркувань: для варіанта а)X4=0,0001, тому що такий варіант підходить для клієнтів не вимогливих до точності розвязку; для варіанта б) X5=0,000001 – висока точність розвязку є необхідною умовою.

Коефіцієнт технічного рівня для кожного варіанта реалізації ПП розраховується за формулою:

$$K_{TP} = \sum_i K_{вi} \cdot B_i, i = 1...n, \quad (5.9)$$

де n – кількість параметрів, $Кв_i$ – коефіцієнт вагомості i–го параметра, B_i – оцінка i–го параметра в балах.

Розраховані показники рівня якості внесемо в таблицю 6. За цими даними визначаємо рівень якості кожного з варіантів:

K_{TEP1} = 0,731+ 0,91+ 1,21+ 0,421 = 3,272

K_{TEP2} = 0,731+ 0,91+ 1,21+ 1,3= 4,151

K_{TEP3} = 0,731 + 0,91+ 0,301+ 0,421= 2,363

K_{TEP4} = 0,731+ 0,91+ 0,301+ 1,3= 3,242

Таблиця 6 – Розрахунок показників рівня якості варіантів реалізації

Основні функції	Варіант реалізації функції	Абсолютне значення параметра	Бальна оцінка параметра	Коефіцієнт вагомості параметра	Коефіцієнт рівня якості
F1	б	11000	4	0,2415254237	0,731
F2	а	16	5	0,1949152542	0,91
F3	а	800	4	0,1949152542	1,21
	б	80	1	0,25	0,301
F4	а	10E-4	5	0,1186440678	0,421
	в	10E-6	10	0,2415254237	1,3

Отже, найкращим є другий варіант, для якого коефіцієнт технічного рівня має найбільше значення.

4.9 Визначення трудомісткості

При визначенні трудомісткості ПП враховують наступні фактори:

- об’єм ПП в умовних машинних командах (УМК);
- складність розроблюваного ПП;
- ступінь новизни розроблюваного ПП;

Склад функцій, реалізованих у даному продукті:

- обробка даних;
- довідка й навчання;
- чисельні методи.

Даний програмний продукт призначений для вже існуючих комп’ютерів й операційних систем, використання типових програмних продуктів не планується.

Оскільки дана система не має жодної з характеристик, які дозволяють віднести її до першої або другої групи складності, віднесемо її до третьої групи складності.

Додаткові ускладнюючі характеристики програмного продукту:

- забезпечення збіжності методу для р ізнорідних даних = 0,07.

Додатковий поправочний коефіцієнт $K_{сл}$ визначається за формулою:

$$K_{сл} = 1 + \Sigma K_i = 1 + 0,07 = 1,07 \quad (10)$$

Об’єм кожної з функцій визначається відповідно “Каталогу функцій ПС ВТ”, для наших функцій підрахунки зведемо у таблицю 7.

Загальний об’єм розроблюваного ПП для кожного з варіантів дорівнює:

- 1 варіант реалізації – 7227,5;
- 2 варіант реалізації – 9754,5;
- 3 варіант реалізації – 7227,5;
- 4 варіант реалізації – 9938.

Встановлюємо норму часу на розробку програмного продукту T_P та загальну трудомісткість T_0:

T_P

T_{p1} – 385 людино-днів;

T_{p2} – 445 людино-днів;

T_{p3} – 385 людино-днів;

T_{p4} – 445 людино-днів.

T_0:

$T_{01} = K_{сл} * T_{p1} = 1,13 * 385 = 435,05$

$T_{02} = K_{сл} * T_{p2} = 1,13 * 445 = 502,85$

$T_{03} = K_{сл} * T_{p3} = 1,13 * 385 = 435,05$

$T_{04} = K_{сл} * T_{p4} = 1,13 * 445 = 502,85$

Таблиця 7 – Визначення об’єму функцій

Номер функції у каталозі	Вміст функції	Відсоток використання у даному програмному продукті			Об’єм функції, УМК	Уточнення об’єму функції за каталогом, УМК		
		1, 3	2	4		1, 3	2	4
109	Забезпечення збіжності методу для різнорідних даних	40	40	40	1085	434	434	434
203	Створення інтерфейсу користувачу	25	35	35	6260	1565	2191	2191
305	Обробка даних	15	25	30	3670	550,5	917,5	1101

403	Обробка таблиць	20	30	30	3690	738	1107	1107
604	Довідка й навчання	10	10	10	4450	445	445	445
701	Чисельні методи	30	40	40	11650	3495	4660	4660
Всього						7227,5	9754,5	9938

По ступені новизни програмний продукт відноситься до третього ступеню, тому значення поправочного коефіцієнта $K_н$, що її враховує, дорівнює 0,7. У зв'язку з тим, що використання типових програмних продуктів не планується, коефіцієнт приймаємо рівним 1,0.

Оскільки загальна трудомісткість 2 і 4 варіантів збігаються, їх можна об'єднати в одну групу. Тоді в таблиці 4.8, позначення I відповідає 1 й 3 варіантам реалізації, а II – 2 й 4 варіантам.

Таблиця 8 – Розрахунок трудомісткості на всіх ступенях розробки програмного продукту

Ступінь	Код ступеня	Трудомісткість розробки ступеня для різних варіантів	
		I	II
Технічне завдання	ТЗ	27,41	31,67
Ескізний проект	ЭП	0	0
Технічний проект	ТП	42,64	49,28
Робочий проект	РП	185,78	214,7
Впровадження	ВН	48,72	56,32
Разом		304,75	351,97

Отже необхідна кількість виконавців всього програмного продукту за рік – дві людини.

4.10 Визначення витрат на розробку і розрахунок вартості програмного продукту

В розробці беруть участь два програмісти з окладом 5650 грн., один фінансовий аналітик з окладом 9250грн. Визначимо зарплату за годину за формулою:

$$С_ч = \frac{М}{T_m \cdot t} \text{ грн}, \quad (11)$$

де М – місячний оклад працівників; T_m – кількість робочих днів тиждень; t – кількість робочих годин в день.

$$С_ч = \frac{5650 + 5650 + 9250}{3 \cdot 21 \cdot 8} = 44{,}88 \text{ грн} \quad (12)$$

Тоді, розрахуємо заробітну плату за формулою (13)

$$С_{ЗП} = С_ч \cdot Т_i \cdot К_Д, \quad (13)$$

де $С_ч$ – величина погодинної оплати праці програміста;

$Т_i$ – трудомісткість відповідного завдання;

$К_Д$ – норматив, який враховує додаткову заробітну плату.

Зарплата розробників за варіантами становить:

I. $С_{ЗП} = 44{,}88 \cdot 1328.64 \cdot 1.2 = 71555.23$ грн.

II. $С_{ЗП} = 44{,}88 \cdot 1345.52 \cdot 1.2 = 72464.58$ грн.

III. $С_{ЗП} = 44{,}88 \cdot 1328.64 \cdot 1.2 = 71555.23$ грн.

IV. $С_{ЗП} = 44{,}88 \cdot 1345.52 \cdot 1.2 = 72464.58$ грн.

Відрахування на єдиний соціальний внесок в залежності від групи професійного ризику (II клас) становить 22%

I. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 71555,23 \cdot 0.22 = 15742,2$ грн.

II. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 72464,58 \cdot 0.22 = 15942,2$ грн.

III. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 71555,23 \cdot 0.22 = 15742,2$ грн.

IV. $С_{ВІД} = С_{ЗП} \cdot 0.22 = 72464,58 \cdot 0.22 = 15942,2$ грн.

Тепер визначимо витрати на оплату однієї машино-години ($С_М$). Так як одна ЕОМ обслуговує одного програміста з окладом 5650 грн., з коефіцієнтом зайнятості 0,2 то для однієї машини отримаємо:

$$С_Г = 12 \cdot М \cdot К_З = 12 \cdot 5650 \cdot 0,2 = 10360 \text{ грн.} \quad (14)$$

З урахуванням додаткової заробітної плати:

$$С_{ЗП} = С_Г \cdot (1 + К_З) = 10360 \cdot (1 + 0.2) = 12630 \text{ грн.} \quad (15)$$

Відрахування на єдиний соціальний внесок:

$$С_{ВІД} = С_{ЗП} \cdot 0,22 = 12630 \cdot 0,22 = 2778,6 \text{ грн.} \quad (16)$$

Амортизаційні відрахування розраховуємо при амортизації 25% та вартості ЕОМ – 8000 грн.

$$С_А = К_{ТМ} \cdot К_А \cdot Ц_{ПР} = 1.15 \cdot 0.25 \cdot 8000 = 2300 \text{ грн.}, \quad (17)$$

де КТМ – коефіцієнт, який враховує витрати на транспортування та монтаж приладу у користувача;

КА – річна норма амортизації;

ЦПР – договірна ціна приладу.

Витрати на ремонт та профілактику розраховуємо як:

$$С_Р = К_{ТМ} \cdot Ц_{ПР} \cdot К_Р = 1.15 \cdot 8000 \cdot 0.05 = 460 \text{ грн.}, \quad (18)$$

де $К_Р$ – відсоток витрат на поточні ремонти.

Ефективний годинний фонд часу ПК за рік розраховуємо за формулою (16):

$$Т_{ЕФ} = (Д_К – Д_В – Д_С – Д_Р) \cdot t_З \cdot К_В = (365 – 104 – 8 – 16) \cdot 8 \cdot 0.9 = =1706.4 \text{ годин}, \quad (19)$$

де $Д_К$ – календарна кількість днів у році;

$Д_В$, $Д_С$ – відповідно кількість вихідних та святкових днів;

$Д_Р$ – кількість днів планових ремонтів устаткування;

t – кількість робочих годин в день;

$К_В$ – коефіцієнт використання приладу у часі протягом зміни.

Витрати на оплату електроенергії розраховуємо за формулою:

$$С_{ЕЛ} = Т_{ЕФ} \cdot N_С \cdot К_З \cdot Ц_{ЕН} = 1706,4 \cdot 0,156 \cdot 0,2436 \cdot 2,7515 = 178,42 \text{ грн.}, \quad (20)$$

де $N_С$ – середньо споживча потужність приладу;

$К_З$ – коефіцієнтом зайнятості приладу;

$Ц_{ЕН}$ – тариф за 1 КВт-годин електроенергії.

Накладні витрати розраховуємо за формулою (5.18):

$$С_Н = Ц_{ПР} \cdot 0.67 = 6360 \cdot 0,67 = 4261,2 \text{ грн.} \quad (21)$$

Тоді, річні експлуатаційні витрати будуть:

$$С_{ЕКС} = С_{ЗП} + С_{ВІД} + С_А + С_Р + С_{ЕЛ} + С_Н \quad (22)$$

$С_{ЕКС}$ = 12630 + 2778,6 + 2300 + 460 +

+129,69 + 4261,2 = 22559,49 грн.

Собівартість однієї машино-години ЕОМ дорівнюватиме:

$$С_{М\text{-}Г} = С_{ЕКС} / Т_{ЕФ} = 22559,49 / 1706,4 = 13,22 \text{ грн/год.} \quad (23)$$

Оскільки в даному випадку всі роботи, які пов‘язані з розробкою програмного продукту ведуться на ЕОМ, витрати на оплату машинного часу, в залежності від обраного варіанта реалізації, складає:

$$С_М = С_{М\text{-}Г} \cdot Т \quad (24)$$

I. $С_М = 13,22 \cdot 1328,64 = 17564,62$ грн.;

II. $С_М = 13,22 \cdot 1345.52 = 17787,77$ грн.;

III. $С_М = 13,22 \cdot 1328,64 = 17564,62$ грн.;

IV. $С_М = 13,22 \cdot 1345.52 = 17787,77$ грн.;

Накладні витрати складають 67% від заробітної плати:

$$С_Н = С_{ЗП} \cdot 0,67 \quad (25)$$

I. $С_Н = 27118,37 \cdot 0,67 = 18169,31$ грн.;

II. $С_Н = 27422 \cdot 0,67 = 18372,74$ грн.;

III. $С_Н = 27118,37 \cdot 0,67 = 18169,31$ грн.;

IV. $С_Н = 27422 \cdot 0,67 = 18372,74$ грн.;

Отже, вартість розробки ПП за варіантами становить:

$$С_{ПП} = С_{ЗП} + С_{ВІД} + С_{М} + С_{Н} \quad (26)$$

I. $С_{ПП} = 71555,23 + 29549,4 + 19012,84 + 18169,31 = 138286,78$ грн.;

II. $С_{ПП} = 72464,58 + 29880 + 19254,39 + 18372,74 = 139971,71$ грн.;

III. $С_{ПП} = 71555,23 + 29549,4 + 19012,84 + 18169,31 = 138286,78$ грн.;

IV. $С_{ПП} = 72464,58 + 29880 + 19254,39 + 18372,74 = 139971,71$ грн.;

4.11 Розрахунок показників економічної ефективності

Для кожного варіанта реалізації функцій ПП коефіцієнт техніко-економічного рівня розраховуєтьсяза формулою (24):

$$К_{ТЕРj} = К_{Кj} / С_{Фj}, \quad (27)$$

$К_{ТЕР1} = 3,272/138286,78 = 0,00002366;$

$К_{ТЕР2} = 4,151/139971,71 = 0,00002965;$

$К_{ТЕР3} = 2,363/138286,78 = 0,00001708;$

$К_{ТЕР4} = 3,242/139971,71 = 0,00002316;$

Як бачимо, найбільш ефективним є другий варіант реалізації програми з коефіцієнтом техніко-економічного рівня $К_{ТЕР2} = 0,00002965$.

Висновки до розділу 4

В результаті виконання економічного розділу були систематизовані і закріплені теоретичні знання в галузі економіки та організації виробництва використанням їх для техніко-економічного обґрунтування розробки методом функціонально-вартісного аналізу.

На основі даних про зміст основних функцій, які повинен реалізувати програмний продукт, були визначені чотири найбільш перспективні варіанти реалізації продукту. Найбільш ефективним виявився перший варіант як простіший та цілком раціональний.

При такому варіанті виконання ПП він матиме достатню універсальність і легкість модифікації.

ВИСНОВКИ

В результаті виконаної роботи було розроблено двошаровий ансамбль моделей стекінг типу. В якості базових моделей першого рівня використано Random Forest класифікатор, Extra Trees класифікатор, AdaBoost класифікатор, Gradient Boosting класифікатор, Support Vector Machine. Для другого шару використано XGBClassifier з бібліотеки XGBoost. Ансамбль моделей було побудовано на основі поведінкових даних користувачів з сайту. Для використання цих даних в якості вхідних змінних була здійснена агрегація даних. Цільовою змінною моделі є дефолт позичальника. Розроблена модель дозволила досягти показника AUC 0,6.

Розроблена модель готова до впровадження в сучасні системи маніпулювання даними в фінансовій галузі застосування, що використовує різні методи передбачення для фільтрації проблемних клієнтів. Перевагою даної роботи є те, що в якості вхідних даних використовується інформація про поведінку користувача на сайті, що не є приватною.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Breiman, L. [2000] Some infinity theory for predictor ensembles, Technical Report 579, Statistics Dept. UCB

2. Breiman, L.: Random forests. Machine Learning 45(1) (2001) 5–32

3. Raviv Y Intrator N Bootstrapping with noise An effective regularization technique Connection Science

4. Thomas G Dietterich, Ensemble Methods in Machine Learning URL: http://web.engr.oregonstate.edu/~tgd/publications/mcs-ensembles.pdf (дата звертання 14.05.2019)

5. Zhi-Hua Zhou, Ensemble Learning URL: https://cs.nju.edu.cn/zhouzh/zhouzh.files/publication/springerEBR09.pdf (дата звернення 14.05.2019)

6. Ensemble methods: bagging, boosting and stacking URL: https://towardsdatascience.com/ensemble-methods-bagging-boosting-and-stacking-c9214a10a205 (дата звернення 14.05.2019)

7. Ensemble Methods to Optimize Machine Learning Models URL: https://hub.packtpub.com/ensemble-methods-optimize-machine-learning-models/ (дата звернення 14.05.2019)

8. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to Boosting. Journal of Computer and System Sciences 55(1) (1997) 119–139

9. Breiman, L.: Bagging predictors. Machine Learning 24(2) (1996) 123–140

10. Wolpert, D.H.: Stacked generalization. Neural Networks 5(2) (1992) 241–260

11. Schapire, R.E.: The strength of weak learnability. Machine Learning 5(2) (1990) 197–227

12. Hansen, L.K., Salamon, P.: Neural network ensembles. IEEE Transactions on Pattern Analysis and Machine Intelligence 12(10) (1990) 993–1001

13. Simplifying the ROC and AUC metrics URL: https://towardsdatascience.com/understanding-the-roc-and-auc-curves-a05b68550b69 (дата звернення 14.05.2019)

14. Understanding the Bias-Variance Tradeoff URL: http://scott.fortmann-roe.com/docs/BiasVariance.html (дата звернення 14.05.2019)

15. Ting, K.M., Witten, I.H.: Issues in stacked generalization. Journal of Artificial Intelligence Research 10 (1999) 271–289

16. Opitz, D., Maclin, R.: Popular ensemble methods: An empirical study. Journal of Artificial Intelligence Research 11 (1999) 169–198

ДОДАТОК А ЛІСТИНГ ПРОГРАМИ

```
import inline
import matplotlib
import pandas as pd
import numpy as np
import re
import sklearn
import xgboost as xgb
import seaborn as sns
import matplotlib.pyplot as plt
# %matplotlib inline
import datetime
import plotly.offline as py
# py.init_notebook_mode(connected=True)
import plotly.graph_objs as go
import plotly.tools as tls

import warnings

from sklearn.metrics import roc_auc_score
from sklearn.model_selection import KFold
sklearn.set_config(assume_finite=True)

base_dir = "/home/kaambaalaa/diploma/"

dataset = pd.read_csv(base_dir + "all.rpt", sep='\t')
dataset1 = pd.read_csv(base_dir + "features_old.csv", error_bad_lines=False, sep=',')
# print(dataset1.iloc[0, :])

# dataset2 = pd.read_excel("resources/fieldids.xlsx", error_bad_lines=False)
print("===============JOIN STARTED==============")

dataset3 = dataset1.merge(dataset, on=['advanceID'], how='outer')
print("===============JOIN FINISHED==============")
```

```
dataset3.to_csv("/home/kaambaalaa/diploma/features_old_j.csv", index=False)
print("===============JOIN SAVED==============")



print('The scikit-learn version is {}.'.format(sklearn.__version__))


warnings.filterwarnings('ignore')
def noNa(a):
    nans = np.isnan(a)
    infs = np.isinf(a)
    a[nans] = 0
    a[infs] = 0
    return a
# Going to use these 5 base models for the stacking
from sklearn.ensemble import (RandomForestClassifier, AdaBoostClassifier,
                              GradientBoostingClassifier, ExtraTreesClassifier)
from sklearn.svm import SVC
# from sklearn.cross_validation import KFold


def features(dataset):
    return noNa(dataset.loc[:,
        ['preferredFieldId', 'preferredChangeType', 'preferredWeekDay', 'countMonTimestamp', 'countTueTimestamp', 'countWedTimestamp',
         'countThuTimestamp', 'countFriTimestamp', 'countSatTimestamp', 'countSunTimestamp', 'maxAvgSpeed',
         'minAvgSpeed', 'avgAvgSpeed', 'maxSdSpeed', 'minSdSpeed', 'avgSdSpeed', 'preferredIsMobile',
         'preferredDevice', 'preferredBrowser', 'preferredSourceType', 'preferredPageId',
         'count121PageId', 'count9PageId', 'countUniqueDeviceId', 'eventCount', 'LoanNum', 'Fails35NoPayIn90']])


# sklearn.set_config(assume_finite=True)


train = features(pd.read_csv("/home/kaambaalaa/diploma/train.csv", sep=',', low_memory=False))


# train['ProductName'] = train['ProductName'].fillna(0)
# train['ProductName'] = train['ProductName'].map( {'Loan*': 1} ).astype(int)


test = features(pd.read_csv("/home/kaambaalaa/diploma/test.csv", sep=',', low_memory=False))


# Store our advanceID for easy access
advanceID = pd.read_csv("/home/kaambaalaa/diploma/test.csv", sep=',', low_memory=False).loc[:,
        ['advanceID']]['advanceID']
```

```
# print(train.head(100))



colormap = plt.cm.RdBu
plt.figure(figsize=(14,12))
plt.title('Pearson Correlation of Features', y=1.05, size=15)
sns.heatmap(train.astype(float).corr(),linewidths=0.1,vmax=1.0,
            square=True, cmap=colormap, linecolor='white', annot=True)
plt.show()

# Some useful parameters which will come in handy later on
ntrain = train.shape[0]
ntest = test.shape[0]
SEED = 0  # for reproducibility
NFOLDS = 5  # set folds for out-of-fold prediction
kf = KFold(n_splits = NFOLDS, random_state = SEED)
# print(kf.split(train))

# Class to extend the Sklearn classifier
class SklearnHelper(object):
    def __init__(self, clf, seed=0, params=None):
        params['random_state'] = seed
        self.clf = clf(**params)

    def train(self, x_train, y_train):
        self.clf.fit(x_train, y_train)

    def predict(self, x):
        return self.clf.predict(x)

    def fit(self, x, y):
        return self.clf.fit(x, y)

    def feature_importances(self, x, y):
        print(self.clf.fit(x, y).feature_importances_)
```

```
# Class to extend XGboost classifer

def get_oof(clf, x_train, y_train, x_test):
    oof_train = np.zeros((ntrain,))
    oof_test = np.zeros((ntest,))
    oof_test_skf = np.empty((NFOLDS, ntest))
    # print(len(kf.split(x_train)))
    for i, (train_index, test_index) in enumerate(kf.split(x_train)):
        if i % 100000 == 0:
            print(str(i) + "       " + str(datetime.datetime.now()))
        x_tr = x_train[train_index]
        y_tr = y_train[train_index]
        x_te = x_train[test_index]


        clf.train(x_tr, y_tr)


        oof_train[test_index] = clf.predict(x_te)
        oof_test_skf[i, :] = clf.predict(x_test)


    oof_test[:] = oof_test_skf.mean(axis=0)
    return oof_train.reshape(-1, 1), oof_test.reshape(-1, 1)


# Put in our parameters for said classifiers
# Random Forest parameters
rf_params = {
    'n_jobs': -1,
    'n_estimators': 500,
     'warm_start': True,
     #'max_features': 0.2,
    'max_depth': 6,
    'min_samples_leaf': 2,
    'max_features' : 'sqrt',
    'verbose': 0
}


# Extra Trees Parameters
et_params = {
```

```
    'n_jobs': -1,
    'n_estimators':500,
    #'max_features': 0.5,
    'max_depth': 8,
    'min_samples_leaf': 2,
    'verbose': 0
}

# AdaBoost parameters
ada_params = {
    'n_estimators': 500,
    'learning_rate' : 0.75
}

# Gradient Boosting parameters
gb_params = {
    'n_estimators': 500,
     #'max_features': 0.2,
    'max_depth': 5,
    'min_samples_leaf': 2,
    'verbose': 0
}

# Support Vector Classifier parameters
svc_params = {
    'kernel' : 'linear',
    'C' : 0.025
    }

# Create 5 objects that represent our 4 models
rf = SklearnHelper(clf=RandomForestClassifier, seed=SEED, params=rf_params)
et = SklearnHelper(clf=ExtraTreesClassifier, seed=SEED, params=et_params)
ada = SklearnHelper(clf=AdaBoostClassifier, seed=SEED, params=ada_params)
gb = SklearnHelper(clf=GradientBoostingClassifier, seed=SEED, params=gb_params)
svc = SklearnHelper(clf=SVC, seed=SEED, params=svc_params)

# Create Numpy arrays of train, test and target ( Survived) dataframes to feed into our models
y_train = train['Fails35NoPayIn90'].ravel()
train = train.drop(['Fails35NoPayIn90'], axis=1)
```

```
test = test.drop(['Fails35NoPayIn90'], axis=1)
x_train = train.values # Creates an array of the train data
x_test = test.values # Creats an array of the test data

##########################################################################################################3plot stuff
cols = train.columns.values


# Scatter plot
def plot_importance_rf(feature_dataframe):
    # Scatter plot
    trace = go.Scatter(
        y=feature_dataframe['Random Forest feature importances'].values,
        x=feature_dataframe['features'].values,
        mode='markers',
        marker=dict(
            sizemode='diameter',
            sizeref=1,
            size=25,
            #       size= feature_dataframe['AdaBoost feature importances'].values,
            # color = np.random.randn(500), #set color equal to a variable
            color=feature_dataframe['Random Forest feature importances'].values,
            colorscale='Portland',
            showscale=True
        ),
        text=feature_dataframe['features'].values
    )
    data = [trace]

    layout = go.Layout(
        autosize=True,
        title='Random Forest Feature Importance',
        hovermode='closest',
        #     xaxis= dict(
        #         title= 'Pop',
```

```
        #         ticklen= 5,
        #         zeroline= False,
        #         gridwidth= 2,
        #     ),
        yaxis=dict(
            title='Feature Importance',
            ticklen=5,
            gridwidth=2
        ),
        showlegend=False
    )
    fig = go.Figure(data=data, layout=layout)
    py.plot(fig, filename='rf')



# Scatter plot
def plot_importance_et(feature_dataframe):
    # Scatter plot
    trace = go.Scatter(
        y=feature_dataframe['Extra Trees  feature importances'].values,
        x=feature_dataframe['features'].values,
        mode='markers',
        marker=dict(
            sizemode='diameter',
            sizeref=1,
            size=25,
            #       size= feature_dataframe['AdaBoost feature importances'].values,
            # color = np.random.randn(500), #set color equal to a variable
            color=feature_dataframe['Extra Trees  feature importances'].values,
            colorscale='Portland',
            showscale=True
        ),
        text=feature_dataframe['features'].values
    )
    data = [trace]

    layout = go.Layout(
        autosize=True,
```

```
        title='Extra Trees Feature Importance',
        hovermode='closest',
        #     xaxis= dict(
        #         title= 'Pop',
        #         ticklen= 5,
        #         zeroline= False,
        #         gridwidth= 2,
        #     ),
        yaxis=dict(
            title='Feature Importance',
            ticklen=5,
            gridwidth=2
        ),
        showlegend=False
    )
    fig = go.Figure(data=data, layout=layout)
    py.plot(fig, filename='et')


def plot_importance_ada(feature_dataframe):
    # Scatter plot
    trace = go.Scatter(
        y=feature_dataframe['AdaBoost feature importances'].values,
        x=feature_dataframe['features'].values,
        mode='markers',
        marker=dict(
            sizemode='diameter',
            sizeref=1,
            size=25,
            #       size= feature_dataframe['AdaBoost feature importances'].values,
            # color = np.random.randn(500), #set color equal to a variable
            color=feature_dataframe['AdaBoost feature importances'].values,
            colorscale='Portland',
            showscale=True
        ),
        text=feature_dataframe['features'].values
    )
    data = [trace]

    layout = go.Layout(
```

```
        autosize=True,
        title='AdaBoost Feature Importance',
        hovermode='closest',
        #     xaxis= dict(
        #         title= 'Pop',
        #         ticklen= 5,
        #         zeroline= False,
        #         gridwidth= 2,
        #     ),
        yaxis=dict(
            title='Feature Importance',
            ticklen=5,
            gridwidth=2
        ),
        showlegend=False
    )
    fig = go.Figure(data=data, layout=layout)
    py.plot(fig, filename='ada')



def plot_importance_gb(feature_dataframe):
    # Scatter plot
    trace = go.Scatter(
        y=feature_dataframe['Gradient Boost feature importances'].values,
        x=feature_dataframe['features'].values,
        mode='markers',
        marker=dict(
            sizemode='diameter',
            sizeref=1,
            size=25,
            #       size= feature_dataframe['AdaBoost feature importances'].values,
            # color = np.random.randn(500), #set color equal to a variable
            color=feature_dataframe['Gradient Boost feature importances'].values,
            colorscale='Portland',
            showscale=True
        ),
        text=feature_dataframe['features'].values
    )
```

```
    data = [trace]

    layout = go.Layout(
        autosize=True,
        title='Gradient Boosting Feature Importance',
        hovermode='closest',
        #     xaxis= dict(
        #         title= 'Pop',
        #         ticklen= 5,
        #         zeroline= False,
        #         gridwidth= 2,
        #     ),
        yaxis=dict(
            title='Feature Importance',
            ticklen=5,
            gridwidth=2
        ),
        showlegend=False
    )
    fig = go.Figure(data=data, layout=layout)
    py.plot(fig, filename='gb')

##############################################################################################3plot stuff

# Create our OOF train and test predictions. These base results will be used as new features
print("Training was started")
cols = train.columns.values

et_oof_train, et_oof_test = get_oof(et, x_train, y_train, x_test)
print("# Extra Trees")
et_feature = et.feature_importances(x_train, y_train)
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
     'Extra Trees  feature importances': et_feature,
    })
plot_importance_et(feature_dataframe)

rf_oof_train, rf_oof_test = get_oof(rf,x_train, y_train, x_test)
print("# Random Forest")
```

```
rf_feature = rf.feature_importances(x_train,y_train)
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
     'Random Forest feature importances': rf_feature,
    })
plot_importance_rf(feature_dataframe)



ada_oof_train, ada_oof_test = get_oof(ada, x_train, y_train, x_test)
print("# AdaBoost")
ada_feature = ada.feature_importances(x_train, y_train)
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
      'AdaBoost feature importances': ada_feature,
    })
plot_importance_ada(feature_dataframe)

gb_oof_train, gb_oof_test = get_oof(gb,x_train, y_train, x_test)
print("# Gradient Boost")
gb_feature = gb.feature_importances(x_train,y_train)
# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
    'Gradient Boost feature importances': gb_feature
    })
plot_importance_gb(feature_dataframe)

# svc_oof_train, svc_oof_test = get_oof(svc,x_train, y_train, x_test)
# print("# Support Vector Classifier")

print("Training is complete")



# Create a dataframe with features
feature_dataframe = pd.DataFrame( {'features': cols,
     'Random Forest feature importances': rf_feature,
     'Extra Trees  feature importances': et_feature,
      'AdaBoost feature importances': ada_feature,
    'Gradient Boost feature importances': gb_feature
    })
```

```
base_predictions_train = pd.DataFrame( {'RandomForest': rf_oof_train.ravel(),
     'ExtraTrees': et_oof_train.ravel(),
     'AdaBoost': ada_oof_train.ravel(),
      'GradientBoost': gb_oof_train.ravel()
    })
base_predictions_train.head()


data = [
    go.Heatmap(
        z= base_predictions_train.astype(float).corr().values ,
        x=base_predictions_train.columns.values,
        y= base_predictions_train.columns.values,
          colorscale='Viridis',
            showscale=True,
            reversescale = True
    )
]
py.plot(data, filename='labelled-heatmap')


x_train = np.concatenate(( et_oof_train, rf_oof_train, ada_oof_train, gb_oof_train), axis=1)
x_test = np.concatenate(( et_oof_test, rf_oof_test, ada_oof_test, gb_oof_test), axis=1)


def print_predict(y_test, y_pred, y_train, y_pred_train):
    # classifier.fit(X_train, y_train)
    # y_pred = classifier.predict(X_test)
    # y_pred_train = classifier.predict(X_train)
    # print(roc_auc_score(y_test, y_pred))
    print("=========y_pred====================")
    print(roc_auc_score(y_pred, y_test))
    print("==========y_pred_train====================")
    print(roc_auc_score(y_pred_train, y_train))


gbm = xgb.XGBClassifier(
    #learning_rate = 0.02,
 n_estimators= 2000,
 max_depth= 4,
 min_child_weight= 2,
 #gamma=1,
```

```
 gamma=0.9,
 subsample=0.8,
 colsample_bytree=0.8,
 objective= 'binary:logistic',
 nthread= -1,
 scale_pos_weight=1).fit(x_train, y_train)
predictions = gbm.predict(x_test)
# y__train = gbm.predict(x_test)
# print_predict(y_test, predictions, y__train)
# Generate Submission File
StackingSubmission = pd.DataFrame({ 'advanceID': advanceID,
                    'Fails35NoPayIn90': predictions })
StackingSubmission.to_csv("StackingSubmission.csv", index=False)
```

ДОДАТОК Б ІЛЮСТРАТИВНИЙ МАТЕРІАЛ

Ансамбль моделей прогнозування повернення кредиту на основі поведінкових даних

Виконав:
Міщук Андрій Романович
Науковий керівник:
доцент Мілявський Юрій Леонідович

Актуальність роботи

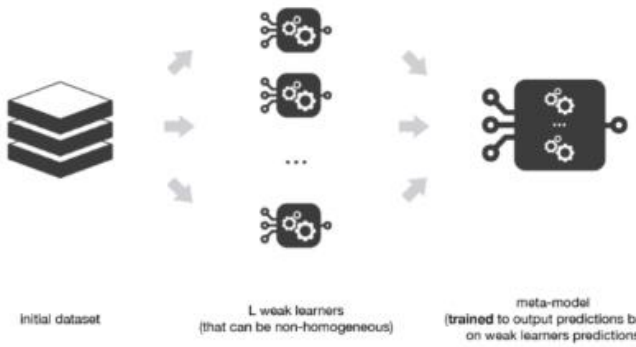
- Покращення прогнозу повернення кредиту для онлайн фінансових установ
- Поведінкові дані є більш доступними, ніж приватні дані клієнтів
- Застосування даної моделі разом з іншими засобами скорингу значно уточнюватимуть якість вибору клієнтів

Постановка задачі

- Вивчення різних методів ансамблювання моделей, засобів підготовки даних та метрик якості
- Вибір алгоритму та побудова базової моделі
- Розробка першого шару, що складається з набору базових моделей
- Обробка результатів моделей першого шару як входів моделі другого шару(ансамбль)

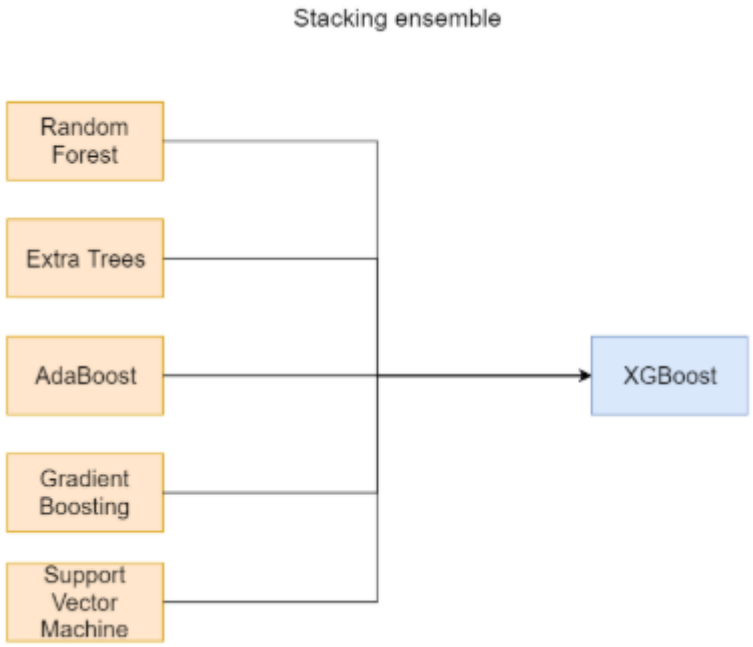
Аналіз найпопулярніших методів ансамблювання

- boosting
- baging
- stacking



Вхідні дані

- поведінкові дані з сайту - логи маніпуляцій користувача
- в якості цільової функції використовується показник прострочення платежів більше ніж 35 днів за перші 90 днів кредиту



Висновки

- досліджено актуальні методи ансамблювання моделей та підготовки даних
- побудовано ансамбль, що має кращі показники, ніж базова модель
- модель цілком готова до інтеграції з джерелом даних

Перспективи

- збагачення даних з допомогою соціальних мереж
- наступні дослідження потребують більшого об’єму даних
- інтеграція моделі в онлайн фінансову компанію шляхом використання технологій Apache Ignite та Apache Kafka